

# Novell Developer Kit

[www.novell.com](http://www.novell.com)

---

NETWARE® CORE PROTOCOLS

June 1, 2005



**Novell®**

## Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

You may not use, export, or re-export this product in violation of any applicable laws or regulations including, without limitation, U.S. export regulations or the laws of the country in which you reside.

Copyright © 1993-2005 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.novell.com/company/legal/patents/> and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.  
404 Wyman Street, Suite 500  
Waltham, MA 02451  
U.S.A.

[www.novell.com](http://www.novell.com)

NDK: NetWare Core Protocols

June 1, 2005

**Online Documentation:** To access the online documentation for this and other Novell developer products, and to get updates, see [developer.novell.com/ndk](http://developer.novell.com/ndk). To access online documentation for Novell products, see [www.novell.com/documentation](http://www.novell.com/documentation).

## Novell Trademarks

AppNotes is a registered trademark of Novell, Inc.

AppTester is a registered trademark of Novell, Inc. in the United States.

ASM is a trademark of Novell, Inc.

BorderManager is a registered trademark of Novell, Inc.

BrainShare is a registered service mark of Novell, Inc. in the United States and other countries.

C3PO is a trademark of Novell, Inc.

Certified Novell Engineer is a service mark of Novell, Inc.

Client32 is a trademark of Novell, Inc.

CNE is a registered service mark of Novell, Inc.

ConsoleOne is a registered trademark of Novell, Inc.

Controlled Access Printer is a trademark of Novell, Inc.

Custom 3rd-Party Object is a trademark of Novell, Inc.

DeveloperNet is a registered trademark of Novell, Inc. in the United States and other countries.

DirXML is a registered trademark of Novell, Inc.

eDirectory is a trademark of Novell, Inc.

Exceleator is a trademark of Novell, Inc.

exteNd is a trademark of Novell, Inc.

exteNd Director is a trademark of Novell, Inc.

exteNd Workbench is a trademark of Novell, Inc.

FAN-OUT FAILOVER is a trademark of Novell, Inc.

GroupWise is a registered trademark of Novell, Inc. in the United States and other countries.

Hardware Specific Module is a trademark of Novell, Inc.

Hot Fix is a trademark of Novell, Inc.

iChain is a registered trademark of Novell, Inc.

Internetwork Packet Exchange is a trademark of Novell, Inc.

IPX is a trademark of Novell, Inc.

IPX/SPX is a trademark of Novell, Inc.

jBroker is a trademark of Novell, Inc.

Link Support Layer is a trademark of Novell, Inc.

LSL is a trademark of Novell, Inc.

ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.

Mirrored Server Link is a trademark of Novell, Inc.

Mono is a registered trademark of Novell, Inc.

MSL is a trademark of Novell, Inc.

My World is a registered trademark of Novell, Inc. in the United States.

NCP is a trademark of Novell, Inc.

NDPS is a registered trademark of Novell, Inc.

NDS is a registered trademark of Novell, Inc. in the United States and other countries.

NDS Manager is a trademark of Novell, Inc.

NE2000 is a trademark of Novell, Inc.

NetMail is a registered trademark of Novell, Inc.

NetWare is a registered trademark of Novell, Inc. in the United States and other countries.

NetWare/IP is a trademark of Novell, Inc.

NetWare Core Protocol is a trademark of Novell, Inc.

NetWare Loadable Module is a trademark of Novell, Inc.

NetWare Management Portal is a trademark of Novell, Inc.

NetWare Name Service is a trademark of Novell, Inc.

NetWare Peripheral Architecture is a trademark of Novell, Inc.

NetWare Requester is a trademark of Novell, Inc.

NetWare SFT and NetWare SFT III are trademarks of Novell, Inc.

NetWare SQL is a trademark of Novell, Inc.

NetWire is a registered service mark of Novell, Inc. in the United States and other countries.

NLM is a trademark of Novell, Inc.

NMAS is a trademark of Novell, Inc.

NMS is a trademark of Novell, Inc.

Novell is a registered trademark of Novell, Inc. in the United States and other countries.

Novell Application Launcher is a trademark of Novell, Inc.

Novell Authorized Service Center is a service mark of Novell, Inc.

Novell Certificate Server is a trademark of Novell, Inc.

Novell Client is a trademark of Novell, Inc.

Novell Cluster Services is a trademark of Novell, Inc.

Novell Directory Services is a registered trademark of Novell, Inc.

Novell Distributed Print Services is a trademark of Novell, Inc.

Novell iFolder is a registered trademark of Novell, Inc.

Novell Labs is a trademark of Novell, Inc.

Novell SecretStore is a registered trademark of Novell, Inc.

Novell Security Attributes is a trademark of Novell, Inc.

Novell Storage Services is a trademark of Novell, Inc.

Novell, Yes, Tested & Approved logo is a trademark of Novell, Inc.

Nsure is a registered trademark of Novell, Inc.

Nterprise is a trademark of Novell, Inc.

Nterprise Branch Office is a trademark of Novell, Inc.

ODI is a trademark of Novell, Inc.

Open Data-Link Interface is a trademark of Novell, Inc.

Packet Burst is a trademark of Novell, Inc.

PartnerNet is a registered service mark of Novell, Inc. in the United States and other countries.

Printer Agent is a trademark of Novell, Inc.

QuickFinder is a trademark of Novell, Inc.

Red Box is a trademark of Novell, Inc.

Red Carpet is a registered trademark of Novell, Inc. in the United States and other countries.

Sequenced Packet Exchange is a trademark of Novell, Inc.

SFT and SFT III are trademarks of Novell, Inc.

SPX is a trademark of Novell, Inc.

Storage Management Services is a trademark of Novell, Inc.

SUSE is a registered trademark of SUSE AG, a Novell business.

System V is a trademark of Novell, Inc.

Topology Specific Module is a trademark of Novell, Inc.

Transaction Tracking System is a trademark of Novell, Inc.

TSM is a trademark of Novell, Inc.

TTS is a trademark of Novell, Inc.

Universal Component System is a registered trademark of Novell, Inc.

Virtual Loadable Module is a trademark of Novell, Inc.

VLM is a trademark of Novell, Inc.

Yes Certified is a trademark of Novell, Inc.

ZENworks is a registered trademark of Novell, Inc. in the United States and other countries.

## **Third-Party Materials**

All third-party trademarks are the property of their respective owners.



# Contents

NetWare Core Protocols Preface	21
<b>Part I NCP Numbers</b>	
<b>1 NCPs By Number</b>	<b>25</b>
17 xx . . . . .	25
21 xx . . . . .	25
22 xx . . . . .	25
23 xx . . . . .	26
32 xx . . . . .	26
34 xx . . . . .	26
35 xx . . . . .	26
36 xx . . . . .	27
86 xx . . . . .	27
87 xx . . . . .	27
89 xx . . . . .	27
90 xx . . . . .	27
104 xx. . . . .	28
111 xx. . . . .	28
114 xx. . . . .	28
123 xx. . . . .	28
131 xx. . . . .	28
<b>2 Return Values</b>	<b>29</b>
RPC Return Values. . . . .	33
<b>Part II Accounting</b>	
<b>3 Concepts</b>	<b>37</b>
Accounting Example . . . . .	37
<b>4 NCPs</b>	<b>39</b>
Get Current Account Status 0x2222 23 150 . . . . .	40
Submit Account Charge 0x2222 23 151 . . . . .	42
Submit Account Hold 0x2222 23 152 . . . . .	44
Submit Account Note 0x2222 23 153 . . . . .	46
<b>Part III AFP</b>	
<b>5 Concepts</b>	<b>51</b>
AFP File Names . . . . .	51
AFP to DOS Conversions . . . . .	51
AFP Path Names . . . . .	52
<b>6 NCPs</b>	<b>53</b>
AFP 2.0 Create Directory 0x2222 35 13 . . . . .	54
AFP 2.0 Create File 0x2222 35 14 . . . . .	56

AFP 2.0 Get File Or Directory Information 0x2222 35 15 . . . . .	58
AFP 2.0 Scan File Information 0x2222 35 17 . . . . .	63
AFP 2.0 Set File Information 0x2222 35 16 . . . . .	68
AFP Alloc Temporary Directory Handle 0x2222 35 11 . . . . .	71
AFP Create Directory 0x2222 35 01 . . . . .	73
AFP Create File 0x2222 35 02 . . . . .	75
AFP Delete 0x2222 35 03 . . . . .	77
AFP Get DOS Name From Entry ID 0x2222 35 18 . . . . .	79
AFP Get Entry ID From Name 0x2222 35 04 . . . . .	81
AFP Get Entry ID From NetWare Handle 0x2222 35 06 . . . . .	83
AFP Get Entry ID From Path Name 0x2222 35 12 . . . . .	84
AFP Get File Information 0x2222 35 05 . . . . .	86
AFP Get Macintosh Info On Deleted File 0x2222 35 19 . . . . .	91
AFP Open File Fork 0x2222 35 08 . . . . .	92
AFP Rename 0x2222 35 07 . . . . .	94
AFP Scan File Information 0x2222 35 10 . . . . .	96
AFP Set File Information 0x2222 35 09 . . . . .	101

## Part IV Bindery

<b>7 Concepts</b>	<b>107</b>
Bindery Objects . . . . .	107
ObjectType . . . . .	107
ObjectName . . . . .	108
ObjectSecurity . . . . .	108
Properties and Values . . . . .	109
PropertyName . . . . .	110
PropertyFlags . . . . .	110
PropertySecurity . . . . .	110
Property List . . . . .	111
Workgroup Managers . . . . .	112
<b>8 NCPs</b>	<b>115</b>
Add Bindery Object To Set 0x2222 23 65 . . . . .	116
Change Bindery Object Password 0x2222 23 64 . . . . .	118
Change Bindery Object Security 0x2222 23 56 . . . . .	120
Change Property Security 0x2222 23 59 . . . . .	122
Change User Password (old) 0x2222 23 01 . . . . .	124
Close Bindery 0x2222 23 68 . . . . .	126
Create Bindery Object 0x2222 23 50 . . . . .	127
Create Property 0x2222 23 57 . . . . .	129
Delete Bindery Object 0x2222 23 51 . . . . .	131
Delete Bindery Object From Set 0x2222 23 66 . . . . .	133
Delete Property 0x2222 23 58 . . . . .	135
Get Bindery Access Level 0x2222 23 70 . . . . .	137
Get Bindery Object Access Level 0x2222 23 72 . . . . .	139
Get Bindery Object ID 0x2222 23 53 . . . . .	140
Get Bindery Object Name 0x2222 23 54 . . . . .	142
Get Group Number (old) 0x2222 23 07 . . . . .	144
Get User Number (old) 0x2222 23 03 . . . . .	146
Is Bindery Object In Set 0x2222 23 67 . . . . .	148
Is Calling Station a Manager 0x2222 23 73 . . . . .	150
Keyed Change Password 0x2222 23 75 . . . . .	151
Keyed Verify Password 0x2222 23 74 . . . . .	152
List Relations Of an Object 0x2222 23 76 . . . . .	153
Open Bindery 0x2222 23 69 . . . . .	155

Read Property Value 0x2222 23 61 . . . . .	156
Rename Object 0x2222 23 52 . . . . .	158
Scan Bindery Object 0x2222 23 55 . . . . .	160
Scan Bindery Object (List) 0x2222 23 32 . . . . .	162
Scan Bindery Object Trustee Paths 0x2222 23 71 . . . . .	164
Scan Property 0x2222 23 60 . . . . .	166
Verify Bindery Object Password 0x2222 23 63 . . . . .	168
Write Property Value 0x2222 23 62 . . . . .	170
<b>9 Structures</b>	<b>173</b>
ObjectInfoStruc . . . . .	174
<b>Part V Connection</b>	
<b>10 Concepts</b>	<b>177</b>
Unreliable Messaging. . . . .	177
Timeout Period . . . . .	177
<b>11 NCPs</b>	<b>179</b>
Change Connection State 0x2222 23 29 . . . . .	180
Clear Connection Number 0x2222 23 254 . . . . .	181
Create Service Connection 0x1111 . . . . .	182
Destroy Service Connection 0x5555 00 . . . . .	185
End Of Job 0x2222 24 . . . . .	187
Generate GUIDs 0x2222 23 33 . . . . .	188
Get Big Packet NCP Max Packet Size 0x2222 97 . . . . .	189
Get Connection List From Object 0x2222 23 31 . . . . .	191
Get Internet Address 0x2222 23 26. . . . .	192
Get Internet Address (old) 0x2222 23 19 . . . . .	193
Get Login Key 0x2222 23 23 . . . . .	195
Get Object Connection List 0x2222 23 27 . . . . .	196
Get Object Connection List (old) 0x2222 23 21. . . . .	197
Get Station Number 0x2222 19. . . . .	199
Get Station's Logged Info 0x2222 23 28 . . . . .	200
Get Station's Logged Info (old) 0x2222 23 22 . . . . .	201
Get Station's Logged Info (old) 0x2222 23 05 . . . . .	203
Get User Connection List (old) 0x2222 23 02. . . . .	205
Keyed Object Login 0x2222 23 24 . . . . .	207
Login Object 0x2222 23 20 . . . . .	208
Login User (old) 0x2222 23 00 . . . . .	210
Logout 0x2222 25. . . . .	212
Negotiate Buffer Size 0x2222 33 . . . . .	213
Request Being Processed 0x9999 . . . . .	215
Request Processed 0x3333. . . . .	216
Set Watchdog Delay Interval 0x2222 23 30 . . . . .	218
Set Connection Language Encoding 0x2222 23 34 . . . . .	219
<b>12 Structures</b>	<b>221</b>
NetworkAddressStruct . . . . .	222
<b>Part VI Data Migration</b>	
<b>13 Concepts</b>	<b>225</b>
Support Modules . . . . .	225
<b>14 NCPs</b>	<b>227</b>
DM File Information 0x2222 90 129. . . . .	228

DM Support Module Information 0x2222 90 132 . . . . .	230
DM Support Module Capacity Request 0x2222 90 135 . . . . .	233
Get/Set Default Read-Write Support Module ID 0x2222 90 134. . . . .	234
Migrator Status Info 0x2222 90 131 . . . . .	236
Move File Data From DM 0x2222 90 133 . . . . .	238
Move File Data To DM 0x2222 90 128 . . . . .	240
RTDM Request 0x2222 90 136 . . . . .	242
Volume DM Status 0x2222 90 130 . . . . .	243
<b>15 Structures</b>	<b>245</b>
DMINFO . . . . .	246
SMCapacityReq . . . . .	247
<b>Part VII Extended Attributes</b>	
<b>16 Concepts</b>	<b>251</b>
Read and Write Positions . . . . .	251
<b>17 NCPs</b>	<b>253</b>
Close Extended Attribute Handle 0x2222 86 01 . . . . .	254
Duplicate Extended Attributes 0x2222 86 05 . . . . .	255
Enumerate Extended Attribute 0x2222 86 04 . . . . .	257
Read Extended Attribute 0x2222 86 03 . . . . .	260
Write Extended Attribute 0x2222 86 02 . . . . .	262
<b>18 Structures</b>	<b>265</b>
ECHandleStruct . . . . .	266
EnumEAStruct_Lvl1 . . . . .	267
EnumEAStruct_Lvl6 . . . . .	268
EnumEAStruct_Lvl7 . . . . .	269
<b>19 Values</b>	<b>271</b>
Access Flag Values . . . . .	271
EA Return Values . . . . .	271
Flags Values. . . . .	272
Multiple Purpose NCP Return Values . . . . .	273
<b>Part VIII File System</b>	
<b>20 Concepts</b>	<b>277</b>
File Security . . . . .	277
File Attributes . . . . .	277
File Names . . . . .	279
Directory Access Rights . . . . .	280
Directory Handle Table . . . . .	281
Workstation Tables . . . . .	282
Directory Tables . . . . .	283
Volume Tables . . . . .	283
NetWare 3.1 and Above . . . . .	283
Name Space Information Bit Mask . . . . .	283
<b>21 NCPs</b>	<b>285</b>
Add Extended Trustee to Directory or File 0x2222 22 39 . . . . .	286
Add Trustee to Directory 0x2222 22 13 . . . . .	287
Add Trustee Set to File or Subdirectory 0x2222 87 10 . . . . .	289
Add User Disk Space Restriction 0x2222 22 33. . . . .	290
Alloc Permanent Directory Handle 0x2222 22 18 . . . . .	291

Alloc Special Temporary Directory Handle 0x2222 22 22 . . . . .	293
Alloc Temporary Directory Handle 0x2222 22 19 . . . . .	295
Allocate Short Directory Handle 0x2222 87 12 . . . . .	297
Close File 0x2222 66 . . . . .	299
Commit File 0x2222 59 . . . . .	300
Convert Path to Dir Entry 0x2222 23 244 . . . . .	301
Copy from One File to Another 0x2222 74 . . . . .	302
Create Directory 0x2222 22 10 . . . . .	304
Create File 0x2222 67 . . . . .	306
Create New File 0x2222 77 . . . . .	308
Deallocate Directory Handle 0x2222 22 20 . . . . .	310
Delete Directory 0x2222 22 11 . . . . .	311
Delete a File or Subdirectory 0x2222 87 08 . . . . .	313
Delete Trustee from Directory 0x2222 22 14 . . . . .	314
Delete Trustee Set from File or SubDirectory 0x2222 87 11 . . . . .	316
Erase File 0x2222 68 . . . . .	317
Extract a Base Handle 0x2222 22 23 . . . . .	319
File Migration Request 0x2222 90 150 . . . . .	320
File Search Continue 0x2222 63 . . . . .	321
File Search Initialize 0x2222 62 . . . . .	324
Generate Directory Base and Volume Number 0x2222 87 22 . . . . .	326
Get Current Size of File 0x2222 87 66 . . . . .	328
Get Current Size of File 0x2222 71 . . . . .	330
Get Directory Disk Space Restriction 0x2222 22 35 . . . . .	332
Get Directory Disk Space Restriction 0x2222 87 39 . . . . .	334
Get Directory Entry 0x2222 22 31 . . . . .	336
Get Directory Information 0x2222 22 45 . . . . .	338
Get Directory Path 0x2222 22 01 . . . . .	340
Get Effective Directory Rights 0x2222 22 03 . . . . .	342
Get Effective Directory Rights 0x2222 87 29 . . . . .	343
Get Effective Rights for Directory Entry 0x2222 22 42 . . . . .	344
Get Extended Volume Information 0x2222 22 51 . . . . .	345
Get File Information 0x2222 87 31 . . . . .	347
Get Full Path String 0x2222 87 28 . . . . .	351
Get Huge NS Information 0x2222 87 26 . . . . .	353
Get Mount Volume List 0x2222 22 52 . . . . .	355
Get NS Information 0x2222 87 19 . . . . .	357
Get Name Space Directory Entry 0x2222 22 48 . . . . .	359
Get Name Space Information 0x2222 22 47 . . . . .	363
Get Name Spaces Loaded List from Volume Number 0x2222 87 24 . . . . .	365
Get Object Disk Usage and Restrictions 0x2222 22 41 . . . . .	366
Get Object Effective Rights for Directory Entry 0x2222 22 50 . . . . .	368
Get Path Name of a Volume-Directory Number Pair 0x2222 22 26 . . . . .	370
Get Path String from Short Directory Handle 0x2222 87 21 . . . . .	371
Get Reference Count from Dir Entry Number 0x2222 90 10 . . . . .	372
Get Reference Count from Dir Handle 0x2222 90 11 . . . . .	373
Get Sparse File Data Block Bit Map 0x2222 85 . . . . .	374
Get Volume and Purge Information 0x2222 22 44 . . . . .	375
Get Volume Info with Handle 0x2222 22 21 . . . . .	377
Get Volume Info with Number 0x2222 18 . . . . .	379
Get Volume Name 0x2222 22 06 . . . . .	381
Get Volume Number 0x2222 22 05 . . . . .	383
Initialize Search 0x2222 87 02 . . . . .	384
Map Directory Number to Path 0x2222 23 243 . . . . .	385
Modify DOS Attributes on a File or Subdirectory 0x2222 87 35 . . . . .	387

Modify File or Subdirectory DOS Information 0x2222 87 07. . . . .	389
Modify Maximum Rights Mask 0x2222 22 04 . . . . .	390
Obtain File or Subdirectory Information 0x2222 87 06. . . . .	392
Open CallBack Control 0x2222 87 34 . . . . .	394
Open/Create File (old) 0x2222 84 . . . . .	395
Open/Create File or Subdirectory 0x2222 87 01. . . . .	398
Open/Create File or Subdirectory 0x2222 87 30. . . . .	401
Open/Create File or Subdirectory with Callback 0x2222 87 32 . . . . .	404
Open/Create File or Subdirectory II with Callback 0x2222 87 33 . . . . .	407
Open/Create File or Subdirectory 0x2222 89 01. . . . .	410
Open Data Stream 0x2222 22 49 . . . . .	413
Open File (old) 0x2222 65 . . . . .	415
Open File 0x2222 76. . . . .	417
Parse Tree 0x2222 90 00 . . . . .	420
Purge Erased Files (old) 0x2222 22 16 . . . . .	422
Purge Salvageable File 0x2222 87 18. . . . .	423
Purge Salvageable File (old) 0x2222 22 29 . . . . .	424
Purge Salvageable File List 0x2222 87 42 . . . . .	425
Query NS Information Format 0x2222 87 23 . . . . .	427
Read File 0x2222 87 64 . . . . .	429
Read From a File 0x2222 72 . . . . .	431
Recover Erased File (old) 0x2222 22 17. . . . .	433
Recover Salvageable File 0x2222 87 17 . . . . .	435
Recover Salvageable File (old) 0x2222 22 28. . . . .	436
Remove Extended Trustee from Dir or File 0x2222 22 43. . . . .	437
Remove User Disk Space Restrictions 0x2222 22 34 . . . . .	438
Rename Directory 0x2222 22 15 . . . . .	439
Rename File 0x2222 69 . . . . .	441
Rename Or Move (old) 0x2222 22 46 . . . . .	443
Rename Or Move a File or Subdirectory 0x2222 87 04 . . . . .	445
Restore an Extracted Base Handle 0x2222 22 24. . . . .	446
Revoke File Handle Rights 0x2222 87 43 . . . . .	448
Update File Handle Rights 0x2222 87 44 . . . . .	450
Scan a Directory 0x2222 22 30 . . . . .	452
Scan Directory Disk Space 0x2222 22 40 . . . . .	455
Scan Directory for Trustees 0x2222 22 12. . . . .	458
Scan Directory Information 0x2222 22 02 . . . . .	460
Scan File Information 0x2222 23 15. . . . .	462
Scan File or Directory for Extended Trustees 0x2222 22 38 . . . . .	466
Scan File or Subdirectory for Trustees 0x2222 87 05 . . . . .	468
Scan Salvageable File List 0x2222 87 41 . . . . .	470
Scan Salvageable Files 0x2222 87 16. . . . .	472
Scan Salvageable Files (old) 0x2222 22 27 . . . . .	474
Scan Volume's User Disk Restrictions 0x2222 22 32 . . . . .	476
Search for a File 0x2222 64 . . . . .	478
Search for File or Subdirectory 0x2222 87 03 . . . . .	480
Search for File or Subdirectory Set 0x2222 87 20. . . . .	482
Search for File or Subdirectory Set (Extended Errors) 0x2222 87 40 . . . . .	484
Set Compressed File Size 0x2222 90 12 . . . . .	486
Set Directory Disk Space Restriction 0x2222 22 36. . . . .	487
Set Directory Entry Information 0x2222 22 37. . . . .	488
Set Directory Handle 0x2222 22 00 . . . . .	491
Set Directory Information 0x2222 22 25 . . . . .	493
Set File Attributes 0x2222 70 . . . . .	495
Set File Extended Attributes 0x2222 79 . . . . .	497

Set File Information 0x2222 23 16 . . . . .	499
Set File Time Date Stamp 0x2222 75. . . . .	501
Set Huge NS Information 0x2222 87 27 . . . . .	502
Set NS Information 0x2222 87 25. . . . .	504
Set Short Directory Handle 0x2222 87 09 . . . . .	505
Write to a File 0x2222 87 65 . . . . .	507
Write to a File 0x2222 73 . . . . .	509

## **22 Enhanced NCPs 511**

Add Trustee Set to File or Subdirectory 0x2222 89 10 . . . . .	512
Allocate Short Directory Handle 0x2222 89 12 . . . . .	513
Delete a File or Subdirectory 0x2222 89 08 . . . . .	515
Delete Trustee Set from File or SubDirectory 0x2222 89 11 . . . . .	516
Enhanced Enumerate Extended Attribute 0x2222 89 54 . . . . .	517
Enhanced Get Object Effective Rights 0x2222 89 50. . . . .	520
Enhanced Read Extended Attribute 0x2222 89 53 . . . . .	521
Enhanced Scan Volume Trustee Object Paths 0x2222 89 71 . . . . .	523
Enhanced Set NS Information 0x2222 89 25. . . . .	525
Enhanced Write Extended Attribute 0x2222 89 52 . . . . .	527
Generate Directory Base and Volume Number 0x2222 89 22 . . . . .	529
Get Directory Disk Space Restriction 0x2222 89 39 . . . . .	531
Get Effective Directory Rights 0x2222 89 29 . . . . .	533
Get Full Path String 0x2222 89 28 . . . . .	534
Get NS Information 0x2222 89 19 . . . . .	536
Initialize Search 0x2222 89 02 . . . . .	538
Modify DOS Attributes on a File or Subdirectory 0x2222 89 35 . . . . .	539
Modify File or Subdirectory DOS Information 0x2222 89 07 . . . . .	541
Obtain File or Subdirectory Information 0x2222 89 06 . . . . .	542
Open/Create File or Subdirectory 0x2222 89 01 . . . . .	544
Open/Create File or Subdirectory 0x2222 89 30 . . . . .	546
Open/Create File or Subdirectory with Callback 0x2222 89 32 . . . . .	549
Open/Create File or Subdirectory II with Callback 0x2222 89 33. . . . .	552
Recover Salvageable File 0x2222 89 17 . . . . .	555
Rename Or Move a File or Subdirectory 0x2222 89 04. . . . .	556
Scan File or Subdirectory for Trustees 0x2222 89 05. . . . .	557
Scan Salvageable Files 0x2222 89 16 . . . . .	559
Search for File or Subdirectory 0x2222 89 03 . . . . .	561
Search for File or Subdirectory Set 0x2222 89 20 . . . . .	563
Search for File or Subdirectory Set (Extended Errors) 0x2222 89 40. . . . .	565
Set Short Directory Handle 0x2222 89 09 . . . . .	567

## **23 Structures 569**

64BitFileSizeStruct . . . . .	570
AllAttrStruc . . . . .	571
AllDirBaseNumStruc . . . . .	572
ArchiveInfoStruct . . . . .	573
AttributesStruct . . . . .	574
CreationInfoStruct. . . . .	575
CreatorStruc . . . . .	576
DataStreamFATInfo. . . . .	577
DataStreamSizeInfo. . . . .	578
DataStreamSizeStruct . . . . .	579
DataStreamSizesStruc . . . . .	580
DirDiskSpaceResList . . . . .	581
DirEntryStruct . . . . .	582
DOSNameStruct . . . . .	583

DOSTimeInformationBlock . . . . .	584
DSSpaceAllocateStruct . . . . .	585
DStreamActual . . . . .	586
DStreamLogical . . . . .	587
DataStreamSizeInfo . . . . .	588
EALInfoStruct . . . . .	589
EffectiveRightsStruct . . . . .	590
EnhNetWareFileNameStruct . . . . .	591
EnhNetWareHandlePathStruct . . . . .	592
FlushTimeStruct . . . . .	594
IDStruct . . . . .	595
InfoBlock . . . . .	596
InfoCCode . . . . .	597
LastAccessedTimeStruct . . . . .	598
LastUpdatedInSecondsStruct . . . . .	599
limb . . . . .	600
MacFinderInfoStruct . . . . .	601
MacTimeStruct . . . . .	602
MaxSpaceStruct . . . . .	603
ModifyDOSInfoStructure . . . . .	604
ModifyInfoStruct . . . . .	606
NameStruct . . . . .	607
NetWareFileNameStruct . . . . .	608
NetWareHandlePathStruct . . . . .	609
NetWareInformationStruct . . . . .	611
NetWareTrusteeStructure . . . . .	612
NSAttributeStruct . . . . .	613
NSInfoStruct . . . . .	614
ParentBaseIDStruct . . . . .	615
PathComponent . . . . .	616
PathCookie . . . . .	617
Pcomponent . . . . .	618
ReferenceCount . . . . .	619
ReferenceIDStruct . . . . .	620
RightsInfoStruct . . . . .	621
ScanInfoFileName . . . . .	622
ScanInfoFileNoFileName . . . . .	623
SiblingCountStruct . . . . .	624
TotalStreamSizeStruct . . . . .	625
TrusteeStruct . . . . .	626
VollInfoStructure . . . . .	627
VolMntStruct . . . . .	631
VolMntStructWithName . . . . .	632

## 24 Values 633

ChangeBits Values . . . . .	633
DataStream Values . . . . .	633
DesiredAccessRights Values . . . . .	633
FileAttributes Values . . . . .	634
InformationMask Values . . . . .	634
InheritedRightsMask Values . . . . .	636
NameSpace Values . . . . .	636
OpenCreateAction Values . . . . .	636
OpenCreateMode Values . . . . .	636
OpenRights Values . . . . .	637
RenameFlag Values . . . . .	637



ReturnInfoMask Values . . . . .	638
Extended ReturnInfoMask Values . . . . .	638
SearchAttributes Values . . . . .	639

## Part IX Message

<b>25 Concepts</b>	<b>643</b>
Message Buffers . . . . .	643
<b>26 NCPs</b>	<b>645</b>
Broadcast To Console 0x2222 21 09 . . . . .	646
Connection Message Control 0x2222 21 12 . . . . .	647
Disable Broadcasts 0x2222 21 02 . . . . .	648
Enable Broadcasts 0x2222 21 03 . . . . .	649
Get Broadcast Message 0x2222 21 11 . . . . .	650
Get Broadcast Message (old) 0x2222 21 01 . . . . .	651
Log Network Message 0x2222 23 13 . . . . .	652
Send Broadcast Message 0x2222 21 10 . . . . .	654
Send Broadcast Message (old) 0x2222 21 00 . . . . .	656

## Part X NCP Extension

<b>27 Concepts</b>	<b>661</b>
NCP Extension 37 . . . . .	661
NCP Extension 36 . . . . .	661
<b>28 NCPs</b>	<b>663</b>
Execute NCP Extension 0x2222 37 . . . . .	664
Get NCP Extension Information (old) 0x2222 36 00 . . . . .	666
Get NCP Extension Maximum Data Size 0x2222 36 01 . . . . .	668
Get NCP Extension Information by Name 0x2222 36 02 . . . . .	669
Get Number of Registered NCP Extensions 0x2222 36 03 . . . . .	671
Get NCP Extension Registered Verbs List 0x2222 36 04 . . . . .	672
Return NCP Extension Information 0x2222 36 05 . . . . .	673
Return NCP Extension Maximum Data Size 0x2222 36 06 . . . . .	675

## Part XI NDS

<b>29 NCPs</b>	<b>679</b>
Clear Statistics 0x2222 104 07 . . . . .	680
Fragment Close 0x2222 104 03 . . . . .	681
Monitor NDS Connection 0x2222 104 05 . . . . .	682
Ping for NDS NCP 0x2222 104 01 . . . . .	683
Reload NDS Software 0x2222 104 08 . . . . .	684
Return Bindery Context 0x2222 104 04 . . . . .	685
Return NDS Statistics 0x2222 104 06 . . . . .	686
Send NDS Fragmented Request/Reply 0x2222 104 02 . . . . .	688
<b>30 Structures</b>	<b>691</b>
Attribute . . . . .	692
<b>31 NDS Attribute Syntaxes</b>	<b>693</b>
Back Link . . . . .	694
Boolean . . . . .	695
Case Exact String . . . . .	696
Case Ignore List . . . . .	697
Case Ignore Strings . . . . .	698
Class Name . . . . .	699

Counter . . . . .	700
Distinguished Name . . . . .	701
Email Address . . . . .	702
Facsimile Telephone Number . . . . .	703
Hold . . . . .	704
Integer . . . . .	705
Interval . . . . .	706
Net Address . . . . .	707
Numeric String . . . . .	708
Object ACL . . . . .	709
Octet List . . . . .	710
Octet String . . . . .	711
Path . . . . .	712
Postal Address . . . . .	713
Printable String . . . . .	714
Replica Pointer . . . . .	715
Stream . . . . .	716
Telephone Number . . . . .	717
Time . . . . .	718
Timestamp . . . . .	719
TypedName . . . . .	720
Unknown . . . . .	721

## Part XII Packet Burst

<b>32 NCPs</b>	<b>725</b>
Packet Burst Connection Request 0x2222 101 . . . . .	726

## Part XIII Print

<b>33 NCPs</b>	<b>733</b>
Close Spool File 0x2222 17 01 . . . . .	734
Create Spool File 0x2222 17 09 . . . . .	736
Get Printer's Queue 0x2222 17 10 . . . . .	738
Get Printer Status 0x2222 17 06 . . . . .	739
Set Spool File Flags 0x2222 17 02 . . . . .	741
Spool A Disk File 0x2222 17 03 . . . . .	744
Write To Spool File 0x2222 17 00 . . . . .	746

## Part XIV Queue

<b>34 Concepts</b>	<b>751</b>
Status Flags . . . . .	751
Return Values . . . . .	751
<b>35 NCPs</b>	<b>753</b>
Abort Servicing Queue Job 0x2222 23 132 . . . . .	754
Abort Servicing Queue Job (old) 0x2222 23 115 . . . . .	755
Attach Queue Server To Queue 0x2222 23 111 . . . . .	757
Change Job Priority 0x2222 23 130 . . . . .	759
Change Queue Job Entry 0x2222 23 123 . . . . .	760
Change Queue Job Entry (old) 0x2222 23 109 . . . . .	761
Change Queue Job Position 0x2222 23 110 . . . . .	763
Change To Client Rights 0x2222 23 133 . . . . .	765
Change To Client Rights (old) 0x2222 23 116 . . . . .	766
Close File And Start Queue Job 0x2222 23 127 . . . . .	768
Close File And Start Queue Job (old) 0x2222 23 105 . . . . .	769

Create Queue 0x2222 23 100 . . . . .	771
Create Queue Job And File 0x2222 23 121 . . . . .	773
Create Queue Job And File (old) 0x2222 23 104 . . . . .	775
Destroy Queue 0x2222 23 101 . . . . .	778
Detach Queue Server From Queue 0x2222 23 112 . . . . .	780
Finish Servicing Queue Job 0x2222 23 131 . . . . .	782
Finish Servicing Queue Job (old) 0x2222 23 114 . . . . .	783
Get Queue Job File Size 0x2222 23 135 . . . . .	785
Get Queue Job File Size (old) 0x2222 23 120 . . . . .	786
Get Queue Job List 0x2222 23 129 . . . . .	788
Get Queue Job List (old) 0x2222 23 107 . . . . .	790
Get Queue Jobs From Form List 0x2222 23 137 . . . . .	792
Move Queue Job From Src Q to Dst Q 0x2222 23 136 . . . . .	793
Read Queue Current Status 0x2222 23 125 . . . . .	794
Read Queue Current Status (old) 0x2222 23 102 . . . . .	796
Read Queue Job Entry 0x2222 23 122 . . . . .	799
Read Queue Job Entry (old) 0x2222 23 108 . . . . .	800
Read Queue Server Current Status 0x2222 23 134 . . . . .	802
Read Queue Server Current Status (old) 0x2222 23 118 . . . . .	803
Remove Job From Queue 0x2222 23 128 . . . . .	805
Remove Job From Queue (old) 0x2222 23 106 . . . . .	806
Restore Queue Server Rights 0x2222 23 117 . . . . .	808
Service Queue Job 0x2222 23 124 . . . . .	810
Service Queue Job (old) 0x2222 23 113 . . . . .	812
Service Queue Job By Form List 0x2222 23 138 . . . . .	815
Set Queue Current Status 0x2222 23 126 . . . . .	819
Set Queue Current Status (old) 0x2222 23 103 . . . . .	820
Set Queue Server Current Status 0x2222 23 119 . . . . .	822

<b>36 Structures</b>	<b>825</b>
JobStruct . . . . .	826

## Part XV RPC

<b>37 NCPs</b>	<b>833</b>
RPC Add Name Space To Volume 0x2222 131 05 . . . . .	834
RPC Dismount Volume 0x2222 131 04 . . . . .	836
RPC Execute NCF File 0x2222 131 07 . . . . .	838
RPC Load an NLM 0x2222 131 01 . . . . .	840
RPC Mount Volume 0x2222 131 03 . . . . .	842
RPC Set Set Command Value 0x2222 131 06 . . . . .	844
RPC Unload an NLM 0x2222 131 02 . . . . .	846

## Part XVI SecretStore

<b>38 NCPs</b>	<b>851</b>
SecretStore Services 0x2222 92 . . . . .	852

## Part XVII Server Environment

<b>39 Concepts</b>	<b>855</b>
File Access Rights . . . . .	855
<b>40 NCPs</b>	<b>857</b>
Check Console Privileges 0x2222 23 200 . . . . .	858
Clear Connection Number 0x2222 23 254 . . . . .	859
Clear Connection Number (old) 0x2222 23 210 . . . . .	860

Disable File Server Login 0x2222 23 203 . . . . .	861
Disable Transaction Tracking 0x2222 23 207 . . . . .	862
Down File Server 0x2222 23 211 . . . . .	863
Enable File Server Login 0x2222 23 204 . . . . .	864
Enable Transaction Tracking 0x2222 23 208 . . . . .	865
Get Connection's Open Files 0x2222 23 235 . . . . .	866
Get Connection's Open Files (old) 0x2222 23 219 . . . . .	869
Get Connection's Semaphores 0x2222 23 241 . . . . .	872
Get Connection's Semaphores (old) 0x2222 23 225 . . . . .	874
Get Connection's Task Information 0x2222 23 234 . . . . .	876
Get Connection Usage Statistics 0x2222 23 229 . . . . .	878
Get Connection Using A File 0x2222 23 236 . . . . .	880
Get Connection Using A File (old) 0x2222 23 220. . . . .	884
Get Disk Channel Statistics 0x2222 23 217 . . . . .	887
Get Disk Utilization 0x2222 23 14 . . . . .	890
Get Drive Mapping Table 0x2222 23 215 . . . . .	892
Get File Server Date And Time 0x2222 20 . . . . .	895
Get File Server Description Strings 0x2222 23 201 . . . . .	897
Get File Server Information 0x2222 23 17 . . . . .	898
Get File Server LAN I/O Statistics 0x2222 23 231. . . . .	902
Get File Server Login Status 0x2222 23 205 . . . . .	907
Get File Server Misc Information 0x2222 23 232 . . . . .	909
Get File System Statistics 0x2222 23 212 . . . . .	912
Get LAN Driver Configuration Information 0x2222 23 227. . . . .	915
Get Logical Record Information 0x2222 23 240 . . . . .	917
Get Logical Record Information (old) 0x2222 23 224 . . . . .	920
Get Logical Records By Connection 0x2222 23 239 . . . . .	923
Get Logical Records By Connection (old) 0x2222 23 223. . . . .	925
Get Network Serial Number 0x2222 23 18 . . . . .	927
Get Object's Remaining Disk Space 0x2222 23 230 . . . . .	928
Get Physical Record Locks By Connection And File 0x2222 23 237 . . . . .	930
Get Physical Record Locks By Connection And File (old) 0x2222 23 221 . . . . .	933
Get Physical Record Locks By File 0x2222 23 238 . . . . .	936
Get Physical Record Locks By File (old) 0x2222 23 222 . . . . .	939
Get Semaphore Information 0x2222 23 242. . . . .	942
Get Semaphore Information (old) 0x2222 23 226 . . . . .	944
Get Transaction Tracking Statistics 0x2222 23 213 . . . . .	946
Get Volume Information 0x2222 23 233 . . . . .	950
Read Disk Cache Statistics 0x2222 23 214 . . . . .	953
Read Physical Disk Statistics 0x2222 23 216 . . . . .	958
Send Console Broadcast 0x2222 23 253 . . . . .	962
Send Console Broadcast (old) 0x2222 23 209 . . . . .	963
Set File Server Date And Time 0x2222 23 202 . . . . .	964
Verify Serialization 0x2222 23 12 . . . . .	966

<b>41 Structures</b>	<b>967</b>
WaitRecord . . . . .	968

## Part XVIII Statistical

<b>42 NCPs</b>	<b>971</b>
Active LAN Board List 0x2222 123 20 . . . . .	972
Active Protocol Stacks 0x2222 123 40 . . . . .	974
CPU Information 0x2222 123 08. . . . .	976
Enumerate Connection Information from Connection List 0x2222 123 16. . . . .	978
Enumerate NCP Service Network Addresses 0x2222 123 17. . . . .	981

Garbage Collection Information 0x2222 123 07 . . . . .	983
Get Active Connection List by Type 0x2222 123 14 . . . . .	985
Get Cache Information 0x2222 123 01 . . . . .	987
Get Compression and Decompression Time and Counts 0x2222 123 72 . . . . .	989
Get Current Compressing File 0x2222 123 70 . . . . .	991
Get Current DeCompressing File Info List 0x2222 123 71 . . . . .	993
Get Directory Cache Information 0x2222 123 12 . . . . .	994
Get File Server Information 0x2222 123 02. . . . .	996
Get General Router and SAP Information 0x2222 123 50 . . . . .	998
Get Known Networks Information 0x2222 123 53 . . . . .	1000
Get Known Servers Information 0x2222 123 56 . . . . .	1002
Get Loaded Media Number 0x2222 123 47. . . . .	1004
Get Media Manager Object Children's List 0x2222 123 32 . . . . .	1006
Get Media Manager Object Information 0x2222 123 30 . . . . .	1009
Get Media Manager Objects List 0x2222 123 31 . . . . .	1011
Get Media Name by Media Number 0x2222 123 46 . . . . .	1014
Get Network Router Information 0x2222 123 51 . . . . .	1016
Get Network Routers Information 0x2222 123 52. . . . .	1018
Get NLM Loaded List 0x2222 123 10. . . . .	1020
Get NLM Resource Tag List 0x2222 123 15 . . . . .	1022
Get Operating System Version Information 0x2222 123 13 . . . . .	1024
Get Protocol Stack Configuration Information 0x2222 123 41 . . . . .	1028
Get Protocol Stack Custom Information 0x2222 123 43 . . . . .	1031
Get Protocol Stack Numbers by LAN Board Number 0x2222 123 45 . . . . .	1033
Get Protocol Stack Numbers by Media Number 0x2222 123 44 . . . . .	1035
Get Protocol Stack Statistics Information 0x2222 123 42. . . . .	1038
Get Server Information 0x2222 123 54 . . . . .	1041
Get Server Set Categories 0x2222 123 61 . . . . .	1043
Get Server Set Commands Information 0x2222 123 60 . . . . .	1045
Get Server Set Commands Information By Name 0x2222 123 62 . . . . .	1047
Get Server Sources Information 0x2222 123 55 . . . . .	1050
Get Volume Information by Level 0x2222 123 34. . . . .	1052
Get Volume Segment List 0x2222 123 33 . . . . .	1054
IPX SPX Information 0x2222 123 06 . . . . .	1056
LAN Common Counters Information 0x2222 123 22 . . . . .	1058
LAN Configuration Information 0x2222 123 21 . . . . .	1061
LAN Custom Counters Information 0x2222 123 23. . . . .	1063
LAN Name Information 0x2222 123 24 . . . . .	1065
LSL Information 0x2222 123 25. . . . .	1067
LSL Logical Board Statistics 0x2222 123 26 . . . . .	1069
MLID Board Information 0x2222 123 27 . . . . .	1071
NetWare File Systems Information 0x2222 123 03. . . . .	1073
NLM Information 0x2222 123 11 . . . . .	1075
Packet Burst Information 0x2222 123 05 . . . . .	1077
User Information 0x2222 123 04 . . . . .	1079
Volume Switch Information 0x2222 123 09. . . . .	1081

## 43 Structures

**1087**

acctngInfo . . . . .	1088
authInfo . . . . .	1089
BoardName . . . . .	1090
CachelInfo . . . . .	1091
CommonLanStruc. . . . .	1093
Counters . . . . .	1095
CPUInformation . . . . .	1096
CustomCntsInfo. . . . .	1098

DirCacheInfo . . . . .	1099
ExtraCacheCnts. . . . .	1101
FileInfoStruct. . . . .	1102
FileServerCounters . . . . .	1103
FileSystemInfo . . . . .	1104
GenericInfoDef. . . . .	1106
IPXInformation . . . . .	1107
KnownRoutes . . . . .	1109
KnownServStruc. . . . .	1110
LANConfigInfo . . . . .	1111
languageInfo . . . . .	1117
lockInfo. . . . .	1118
LSLInformation. . . . .	1119
MemoryCounters . . . . .	1122
MLIDBoardInfo. . . . .	1123
nameInfo. . . . .	1124
ncpNetworkAddress . . . . .	1125
netAddr . . . . .	1126
NLMInformation . . . . .	1127
PacketBurstInformation . . . . .	1130
printInfo . . . . .	1132
RoutersInfo . . . . .	1134
RTagStructure . . . . .	1135
Segments . . . . .	1136
ServerInfo . . . . .	1137
ServersSrcInfo . . . . .	1140
SPXInformation . . . . .	1141
StackInfo. . . . .	1144
statsInfo . . . . .	1145
timeInfo . . . . .	1146
TrendCounters . . . . .	1147
UserInformation . . . . .	1148
VollInfoDef . . . . .	1151
VollInfo2Def . . . . .	1155

## Part XIX Synchronization

<b>44 Concepts</b>	<b>1159</b>
File and Record Locks. . . . .	1159
Semaphores. . . . .	1159
<b>45 NCPs</b>	<b>1161</b>
Clear File 0x2222 87 38 . . . . .	1162
Clear File (old) 0x2222 07 . . . . .	1163
Clear File Set 0x2222 08 . . . . .	1164
Clear Lock Wait Node 0x2222 112 . . . . .	1165
Clear Logical Record 0x2222 11. . . . .	1166
Clear Logical Record Set 0x2222 14 . . . . .	1167
Clear Physical Record 0x2222 87 69 . . . . .	1168
Clear Physical Record 0x2222 30 . . . . .	1170
Clear Physical Record Set 0x2222 31 . . . . .	1171
Close Semaphore 0x2222 111 04 . . . . .	1172
Close Semaphore (old) 0x2222 32 04 . . . . .	1173
Examine Semaphore 0x2222 111 01 . . . . .	1174
Examine Semaphore (old) 0x2222 32 01 . . . . .	1175
File Set Lock (old) 0x2222 01 . . . . .	1176

File Release Lock 0x2222 02 . . . . .	1177
Lock File Set 0x2222 106 . . . . .	1178
Lock File Set (old) 0x2222 04 . . . . .	1179
Lock Logical Record Set 0x2222 108 . . . . .	1180
Lock Logical Record Set (old) 0x2222 10 . . . . .	1181
Lock Physical Record Set 0x2222 110 . . . . .	1182
Lock Physical Record Set (old) 0x2222 27 . . . . .	1183
Log File 0x2222 87 36 . . . . .	1184
Log File (old) 0x2222 105 . . . . .	1185
Log File (old) 0x2222 03 . . . . .	1186
Log Logical Record 0x2222 107 . . . . .	1187
Log Logical Record (old) 0x2222 09 . . . . .	1188
Log Physical Record 0x2222 87 67 . . . . .	1189
Log Physical Record 0x2222 109 . . . . .	1191
Log Physical Record (old) 0x2222 26 . . . . .	1193
Open Semaphore (old) 0x2222 32 00 . . . . .	1195
Open/Create a Semaphore 0x2222 111 00 . . . . .	1197
Release File 0x2222 87 37 . . . . .	1199
Release File (old) 0x2222 05 . . . . .	1200
Release File Set 0x2222 06 . . . . .	1201
Release Logical Record 0x2222 12 . . . . .	1202
Release Logical Record Set 0x2222 13 . . . . .	1203
Release Physical Record 0x2222 87 68 . . . . .	1204
Release Physical Record 0x2222 28 . . . . .	1206
Release Physical Record Set 0x2222 29 . . . . .	1207
Signal Semaphore (old) 0x2222 32 03 . . . . .	1208
Signal (V) Semaphore 0x2222 111 03 . . . . .	1209
Wait On Semaphore (old) 0x2222 32 02 . . . . .	1210
Wait On (P) Semaphore 0x2222 111 02 . . . . .	1211

<b>46 Structures</b>	<b>1213</b>
ASyncLockWaitStruct . . . . .	1214

## Part XX Time Synchronization

<b>47 NCPs</b>	<b>1217</b>
Timesync Exchange Time 0x2222 114 02 . . . . .	1218
Timesync Get Server List 0x2222 114 05 . . . . .	1220
Timesync Get Time 0x2222 114 01 . . . . .	1221
Timesync Get Version 0x2222 114 12 . . . . .	1223
Timesync Set Server List 0x2222 114 06 . . . . .	1224

<b>48 Structures</b>	<b>1225</b>
Exchange_Time_Struct . . . . .	1226

## Part XXI TTS

<b>49 Concepts</b>	<b>1229</b>
Implicit vs. Explicit Tracking . . . . .	1229
Transaction Process . . . . .	1229
Transaction Monitoring . . . . .	1230

<b>50 NCPs</b>	<b>1231</b>
TTS Abort Transaction 0x2222 34 03 . . . . .	1232
TTS Begin Transaction 0x2222 34 01 . . . . .	1233
TTS End Transaction 0x2222 34 02 . . . . .	1234
TTS Get Application Thresholds 0x2222 34 05 . . . . .	1236

TTS Get Transaction Bits 0x2222 34 09 . . . . .	1238
TTS Get Workstation Thresholds 0x2222 34 07 . . . . .	1239
TTS Is Available 0x2222 34 00 . . . . .	1241
TTS Set Application Thresholds 0x2222 34 06 . . . . .	1242
TTS Set Transaction Bits 0x2222 34 10 . . . . .	1244
TTS Set Workstation Thresholds 0x2222 34 08 . . . . .	1245
TTS Transaction Status 0x2222 34 04 . . . . .	1247

<b>Revision History</b>	<b>1249</b>
-------------------------	-------------



# NetWare Core Protocols Preface

NetWare Core Protocols (NCPs) are the set of routines and primitives that speak the fundamental language of and drive NetWare and make up of the core protocol that NetWare speaks. Each routine is also referred to individually as an NCP. This section describes NCPs in general, explains what the fundamental NCP structures are, and details the way NCPs interface with and provide services to a client.

Terms of Use: Supplement to [Novell Developer Kit License Agreement \(http://developer.novell.com/ndk/license.htm\)](http://developer.novell.com/ndk/license.htm)

The NetWare Core Protocols documentation (NCP Documentation) is subject to the Novell Developer Kit License Agreement and this Supplement.

1. *You acknowledge and agree that only experienced NetWare developers should use the NCP Documentation since it provides access to the underlying NetWare OS. You agree to exercise great care in your use of the NCP Documentation since improper use can cause NetWare server abends.*
2. You may use the NCP Documentation only for providing technical support services to end users of Novell products and to support Your development of Derivative Software that does not: a) enable more than one end user per copy of the Derivative Software to access a NetWare server; or, b) provide NetWare server functions.
3. You agree not to assert a claim against Novell, its licensees, or customers for infringement of a patent resulting from Your use of the NCP Documentation.
4. The NCP Documentation was originally intended for internal use and was not documented in Novell's publicly available software development kits. The NCP Documentation: a) only applies to the versions of the corresponding Novell products specified in the NCP Documentation itself; and, b) may and will be changed by Novell without warning to You in later versions of the corresponding Novell products.





## NCP Numbers

In this section, you can access NCPs by number or figure out what an NCP return value means by using one of the following:

- ♦ [Chapter 1, “NCPs By Number,” on page 25](#)
- ♦ [Chapter 2, “Return Values,” on page 29](#)



# 1

## NCPs By Number

Click one of the following numbers to access the NCP lists for each function number grouping or to access that specific NCP by number.

01	02	03	04	05	06	07	08	09	10
11	12	13	14	17 xx	18	19	20	21 xx	22 xx
23 xx	24	25	26	27	28	29	30	31	32 xx
33	34 xx	35 xx	36 xx	37	59	62	63	64	65
66	67	68	69	70	71	72	73	74	75
76	77	79	84	85	86 xx	87 xx	89 xx	90 xx	92
97	99	100	101	102	103	104 xx	105	106	107
108	109	110	111 xx	114 xx	123 xx	131 xx			

### 17 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	06	09	10
----	----	----	----	----	----	----

### 21 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	09	10	11	12
----	----	----	----	----	----	----	----

### 22 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	04	05	06	10	11	12
13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42

43	44	45	46	47	48	49	50	51	52
----	----	----	----	----	----	----	----	----	----

## 23 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	05	07	12	13	14	15
16	17	18	19	20	21	22	23	24	26
27	28	29	30	31	32	33	34	50	51
52	53	54	55	56	57	58	59	60	61
62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	100	101	102	103	104
105	106	107	108	109	110	111	112	113	114
115	116	117	118	119	120	121	122	123	124
125	126	127	128	129	130	131	132	133	134
135	136	137	138	150	151	152	153	200	201
202	203	204	205	207	208	209	210	211	212
213	214	215	216	217	219	220	221	222	223
224	225	226	227	229	230	231	232	233	234
235	236	237	238	239	240	241	242	243	244
253	254								

## 32 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	04
----	----	----	----	----

## 34 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	04	05	06	07	08	09	10
----	----	----	----	----	----	----	----	----	----	----

## 35 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06	07	08	09	10
11	12	13	14	15	16	17	18	19	

## 36 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	04	05	06
----	----	----	----	----	----	----

## 86 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05
----	----	----	----	----

## 87 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06	07	08	09	10
11	12	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43
44		64	65	66	67	68	69		

## 89 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06
07	08	09	10	11	12
16	17	19	20	22	25
28	29	30	32	33	35
39	40	50	52	53	54
71					

## 90 xx

Click one of the following numbers to access that specific NCP number.

00	10	11	12	128	129	130
131	132	133	134	135	136	

## 104 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06	07	08
----	----	----	----	----	----	----	----

## 111 xx

Click one of the following numbers to access that specific NCP number.

00	01	02	03	04
----	----	----	----	----

## 114 xx

Click one of the following numbers to access that specific NCP number.

01	02	05	06	12
----	----	----	----	----

## 123 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06	07	08	09	10
11	12	13	14	15	16	17	20	21	22
23	24	25	26	27	30	31	32	33	34
40	41	42	43	44	45	46	47	50	51
52	53	54	55	56	60	61	62	70	71
72									

## 131 xx

Click one of the following numbers to access that specific NCP number.

01	02	03	04	05	06	07
----	----	----	----	----	----	----



# 2

## Return Values

The following list comes from the server's errors.h file (see also [“RPC Return Values” on page 33](#)).

Decimal Value	Hex Value	Name
115	0x73	ERR_REVOKE_HANDLE_RIGHTS_NOT_FOUND
116	0x74	ERR_REMOTE_NOT_ALLOWED
117	0x75	ERR_UNKNWON_SUCCESS_OR_FAILURE
118	0x76	ERR_BUFFER_NOT_LONG_ALIGNED
119	0x77	ERR_BUFFER_TOO_SMALL
120	0x78	ERR_VOLUME_FLAG_NOT_SET
121	0x79	ERR_NO_ITEMS_FOUND
122	0x7a	ERR_CONNECTION_ALREADY_TEMPORARY
123	0x7b	ERR_CONNECTION_ALREADY_LOGGED_IN
124	0x7c	ERR_CONNECTION_NOT_AUTHENTICATED
125	0x7d	ERR_CONNECTION_NOT_LOGGED_IN
126	0x7e	ERR_NCP_BOUNDARY_CHECK_FAILED
127	0x7f	ERR_LOCK_WAITING
128	0x80	ERR_LOCK_FAIL
129	0x81	ERR_OUT_OF_HANDLES
130	0x82	ERR_NO_OPEN_PRIVILEGE
131	0x83	ERR_HARD_IO_ERROR
132	0x84	ERR_NO_CREATE_PRIVILEGE
133	0x85	ERR_NO_CREATE_DELETE_PRIVILEGE
134	0x86	ERR_R_O_CREATE_FILE
135	0x87	ERR_CREATE_FILE_INVALID_NAME
136	0x88	ERR_INVALID_FILE_HANDLE

Decimal Value	Hex Value	Name
137	0x89	ERR_NO_SEARCH_PRIVILEGE
138	0x8a	ERR_NO_DELETE_PRIVILEGE
139	0x8b	ERR_NO_RENAME_PRIVILEGE
140	0x8c	ERR_NO_SET_PRIVILEGE
141	0x8d	ERR_SOME_FILES_IN_USE
142	0x8e	ERR_ALL_FILES_IN_USE
143	0x8f	ERR_SOME_READ_ONLY
144	0x90	ERR_ALL_READ_ONLY
145	0x91	ERR_SOME_NAMES_EXIST
146	0x92	ERR_ALL_NAMES_EXIST
147	0x93	ERR_NO_READ_PRIVILEGE
148	0x94	ERR_NO_WRITE_PRIVILEGE
149	0x95	ERR_FILE_DETACHED
150	0x96	ERR_NO_ALLOC_SPACE
150	0x96	ERR_TARGET_NOT_A_SUBDIRECTORY
151	0x97	ERR_NO_SPOOL_SPACE
152	0x98	ERR_INVALID_VOLUME
153	0x99	ERR_DIRECTORY_FULL
154	0x9a	ERR_RENAME_ACROSS_VOLUME
155	0x9b	ERR_BAD_DIR_HANDLE
156	0x9c	ERR_INVALID_PATH
156	0x9c	ERR_NO_SUCH_EXTENSION
157	0x9d	ERR_NO_DIR_HANDLES
158	0x9e	ERR_BAD_FILE_NAME
159	0x9f	ERR_DIRECTORY_ACTIVE
160	0xa0	ERR_DIRECTORY_NOT_EMPTY
161	0xa1	ERR_DIRECTORY_IO_ERROR
162	0xa2	ERR_IO_LOCKED
163	0xa3	ERR_TRANSACTION_RESTARTED
164	0xa4	ERR_RENAME_DIR_INVALID

Decimal Value	Hex Value	Name
165	0xa5	ERR_INVALID_OPENCREATE_MODE
166	0xa6	ERR_ALREADY_IN_USE
167	0xa7	ERR_INVALID_RESOURCE_TAG
168	0xa8	ERR_ACCESS_DENIED
190	0xbe	ERR_INVALID_DATA_STREAM
191	0xbf	ERR_INVALID_NAME_SPACE
192	0xc0	ERR_NO_ACCOUNTING_PRIVILEGES
193	0xc1	ERR_NO_ACCOUNT_BALANCE
194	0xc2	ERR_CREDIT_LIMIT_EXCEEDED
195	0xc3	ERR_TOO_MANY_HOLDS
196	0xc4	ERR_ACCOUNTING_DISABLED
197	0xc5	ERR_LOGIN_LOCKOUT
198	0xc6	ERR_NO_CONSOLE_RIGHTS
208	0xd0	ERR_Q_IO_FAILURE
209	0xd1	ERR_NO_QUEUE
210	0xd2	ERR_NO_Q_SERVER
211	0xd3	ERR_NO_Q_RIGHTS
212	0xd4	ERR_Q_FULL
213	0xd5	ERR_NO_Q_JOB
214	0xd6	ERR_NO_Q_JOB_RIGHTS
214	0xd6	ERR_UNENCRYPTED_NOT_ALLOWED
215	0xd7	ERR_Q_IN_SERVICE
215	0xd7	ERR_DUPLICATE_PASSWORD
216	0xd8	ERR_Q_NOT_ACTIVE
216	0xd8	ERR_PASSWORD_TOO_SHORT
217	0xd9	ERR_Q_STN_NOT_SERVER
217	0xd9	ERR_MAXIMUM_LOGINS_EXCEEDED
218	0xda	ERR_Q_HALTED
218	0xda	ERR_BAD_LOGIN_TIME
219	0xdb	ERR_Q_MAX_SERVERS

Decimal Value	Hex Value	Name
219	0xdb	ERR_NODE_ADDRESS_VIOLATION
220	0xdc	ERR_LOG_ACCOUNT_EXPIRED
222	0xde	ERR_BAD_PASSWORD
223	0xdf	ERR_PASSWORD_EXPIRED
224	0xe0	ERR_NO_LOGIN_CONNECTIONS_AVAILABLE
232	0xe8	ERR_WRITE_TO_GROUP_PROPERTY
233	0xe9	ERR_MEMBER_ALREADY_EXISTS
234	0xea	ERR_NO_SUCH_MEMBER
235	0xeb	ERR_PROPERTY_NOT_GROUP
236	0xec	ERR_NO_SUCH_VALUE_SET
237	0xed	ERR_PROPERTY_ALREADY_EXISTS
238	0xee	ERR_OBJECT_ALREADY_EXISTS
239	0xef	ERR_ILLEGAL_NAME
240	0xf0	ERR_ILLEGAL_WILDCARD
241	0xf1	ERR_BINDERY_SECURITY
242	0xf2	ERR_NO_OBJECT_READ_RIGHTS
243	0xf3	ERR_NO_OBJECT_RENAME_RIGHTS
244	0xf4	ERR_NO_OBJECT_DELETE_RIGHTS
245	0xf5	ERR_NO_OBJECT_CREATE_RIGHTS
246	0xf6	ERR_NO_PROPERTY_DELETE_RIGHTS
247	0xf7	ERR_NO_PROPERTY_CREATE_RIGHTS
248	0xf8	ERR_NO_PROPERTY_WRITE_RIGHTS
249	0xf9	ERR_NO_PROPERTY_READ_RIGHTS
250	0xfa	ERR_TEMP_REMAP
251	0xfb	ERR_UNKNOWN_REQUEST
251	0xfb	ERR_NO_SUCH_PROPERTY
252	0xfc	ERR_MESSAGE_QUEUE_FULL
252	0xfc	ERR_TARGET_ALREADY_HAS_MESSAGE
252	0xfc	ERR_NO_SUCH_OBJECT
253	0xfd	ERR_BAD_STATION_NUMBER

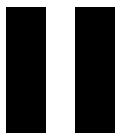
Decimal Value	Hex Value	Name
254	0xfe	ERR_BINDERY_LOCKED
254	0xfe	ERR_DIR_LOCKED
254	0xfe	ERR_SPOOL_DELETE
254	0xfe	ERR_TRUSTEE_NOT_FOUND
255	0xff	ERR_HARD_FAILURE
255	0xff	ERR_FILE_NAME
255	0xff	ERR_FILE_EXISTS
255	0xff	ERR_CLOSE_FCB
255	0xff	ERR_IO_BOUND
255	0xff	ERR_NO_SPOOL_FILE
255	0xff	ERR_BAD_SPOOL_PRINTER
255	0xff	ERR_BAD_PARAMETER
255	0xff	ERR_NO_FILES_FOUND
255	0xff	ERR_NO_TRUSTEE_CHANGE_PRIVILEGE
255	0xff	ERR_TARGET_NOT_LOGGED_IN
255	0xff	ERR_TARGET_NOT_ACCEPTING_MESSAGES
255	0xff	ERR_MUST_FORCE_DOWN
255	0xff	ERR_CHECKSUM_REQUIRED
256	0x100	ERR_SERVICE_ALREADY_LOADED
257	0x101	ERR_SERVICE_NOT_LOADED
258	0x102	ERR_INCORRECT_VERSION
259	0x103	ERR_NO_MEMORY_READ_ACCESS

## RPC Return Values

The following values are returned in a LONG.

Decimal Value	Hex Value	Name
513	0x201	ERR_MODULE_NOT_UNLOADED
514	0x202	ERR_MODULE_NOT_LOADED
515	0x203	ERR_UNABLE_TO_MOUNT_VOLUME

Decimal Value	Hex Value	Name
516	0x204	ERR_UNABLE_TO_DISMOUNT_VOLUME
517	0x205	ERR_UNABLE_TO_ADD_NAME_SPACE
518	0x206	ERR_UNABLE_TO_SET_PARAMETER_VALUE
519	0x207	ERR_UNABLE_TO_EXECUTE_NCF_FILE



## Accounting

Accounting NCPs enable you to create servers that can charge for their services. For example, a database server can charge for the number of records viewed, the number of requests serviced, or the amount of connect time. Likewise, a print server can charge for the number of pages printed.

To access Accounting NCPs, use the following:

- ♦ [Chapter 3, “Concepts,” on page 37](#)
- ♦ [Chapter 4, “NCPs,” on page 39](#)





# 3

## Concepts

This section contains ideas that are common to Accounting NCPs.

### Accounting Example

The following example illustrates the use of accounting system NCPs. For this example, file server FS1 is one of many file servers on an internetwork. FS1's bindery includes several objects, all with various properties, which include the following:

Object Name	Object Type	Object ID	Properties
FS1	FILE_SERVER	00030011h	ACCOUNT_SERVERS. . .
BILL	USER	00060025h	ACCOUNT_BALANCE ACCOUNT_HOLDS. . .
PSERVER	PRINT_SERVER	5C2701F1h	. . .

PSERVER is a print server that advertises its services on the internetwork and appears in FS1's bindery as a dynamic object.

FS1's supervisor declared PSERVER as an accounting server by adding PSERVER's object ID to the 128-byte value segment associated with FS1's ACCOUNT\_SERVERS property. Now, whenever an object in FS1's bindery uses PSERVER to print pages, PSERVER can charge the object (usually a user) for the service.

When a user at workstation WS1 logs in to FS1 as BILL, BILL's ACCOUNT\_BALANCE property has the following 128-byte value segment associated with it:

Offset	Type	Content
0	byte [4]	Balance specifies the amount of money that BILL can use to purchase services on the internetwork. The administrator must choose which monetary unit to use. In this case, each unit equals one cent, so assume BILL's account balance equals \$50.00 (1388h).
4	byte [4]	Credit Limit specifies a fraction of the balance. If the balance falls below the credit limit, BILL can no longer purchase services on the internetwork.
8	byte [120]	Reserved

BILL's ACCOUNT\_HOLDS property has the following 128-byte value segment associated with it:

Offset	Type	Content
0	byte [4]	HoldingServer1: Object ID specifies the object ID of an accounting server that is about to perform some service for BILL.
4	byte [4]	HoldingServer1: Amount specifies the estimated charge for the service. FS1 actually subtracts this amount from BILL's account balance temporarily. No more than 16 accounting servers can appear in ACCOUNT_HOLDS at one time.
...		
120	byte [4]	HoldingServer16: Object ID
124	byte [4]	HoldingServer16: Amount

If BILL makes a request to queue a 10-page file to print on the PSERVER print server, PSERVER completes the following steps:

- ◆ calls **Get Current Account Status 0x2222 23 150 (page 40)** to return BILL's current account status
- ◆ estimates that, at 7 cents per page, BILL's job will cost 70 cents
- ◆ calls **Submit Account Hold 0x2222 23 152 (page 44)** to reserve 70 cents of BILL's account balance
- ◆ queues the print job for printing
- ◆ charges BILL 70 cents for the print job and cancels the 70-cent hold on BILL's account
- ◆ calls **Submit Account Note 0x2222 23 153 (page 46)** to record a note about the purchase and resulting charge in the NET\$ACCT.DAT file located in FS1's SYS:SYSTEM directory

# 4

## NCPs

This section describes each of the Accounting NCPs, their Request and Reply formats, and Return Values.

# Get Current Account Status 0x2222 23 150

Allows a server to get an account status at the time of the request.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (150)	byte
10	ClientType	word (Hi-Lo)
12	ClientNameLen	byte
13	ClientName	byte[ClientNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	AccountBalance	long (Hi-Lo)
12	CreditLimit	long (Hi-Lo)
16	Reserved	byte[120]
136	HolderID1	long (Hi-Lo)
140	HoldAmount1	long (Hi-Lo)
...		
256	HolderID16	long (Hi-Lo)
260	HoldAmount16	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory

Decimal	Hex	Description
192	0xC0	No Account Privileges
193	0xC1	No Account Balance
196	0xC4	Account Disabled
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

No audit record is generated by **Get Current Account Status**.

The reply fields are the same as those described for the ACCOUNT\_BALANCE and ACCOUNT\_HOLDS properties in the “[Accounting Example](#)” on page 37.

*SubFuncStrucLen* = 4 + *ClientNameLen*

If the requesting server’s object ID is not listed in the ACCOUNT\_SERVERS property of the file server’s object, No Account Privileges (192) is returned.

If the requesting (user) object has not ACCOUNT\_BALANCE property, No Account Balance (193) is returned.

# Submit Account Charge 0x2222 23 151

Allows a server to submit a charge to an account.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (151)	byte
10	ServiceType	word (Hi-Lo)
12	ChargeAmount	long (Hi-Lo)
16	HoldCancelAmount	long (Hi-Lo)
20	ClientType	word (Hi-Lo)
22	CommentType	word (Hi-Lo)
24	ClientNameLen	byte
25	ClientName	byte[ClientNameLen]
25+ClientNameLen	CommentLen	byte
26+ClientNameLen	Comment	byte[CommentLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
136	0x88	Invalid File Handle
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
162	0xA2	I/O Lock Error
192	0xC0	No Account Privileges
193	0xC1	No Account Balance

Decimal	Hex	Description
194	0xC2	Credit Limit Exceeded
196	0xC4	Account Disabled
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

**Submit Account Charge** subtracts the charge from the object's account balance, and generates an audit record when the ACCOUNT\_SERVERS property of the file server object exists. The audit record is generated even if a nonzero completion code is returned.

*HoldCancelAmount* is the same amount specified in the previous corresponding call to **Submit Account Hold**. If **Submit Account Hold** was not called prior to providing the service, *HoldCancelAmount* is zero, which has no effect on other outstanding holds by the server.

$SubFuncStrucLen = 17 + ClientNameLen + CommentLen$

If the requesting server is not listed in the ACCOUNT\_SERVERS property of the file server's object, No Account Privileges (192) is returned. If the object specified has not ACCOUNT\_BALANCE property, No Account Balance (193) is returned. In either case, the account will not be charged, even though an entry is recorded in the audit file.

If the object's balance falls below its lowest permissible limit, Credit Limit Exceeded (194) is returned.

# Submit Account Hold 0x2222 23 152

Allows a server to hold an amount, pending a subsequent charge to the client's account.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (152)	byte
10	Amount	long (Hi-Lo)
14	ClientType	word (Hi-Lo)
16	ClientNameLen	byte
17	ClientName	byte[ClientNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
136	0x88	Invalid File Handle
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
162	0xA2	I/O Lock Error
192	0xC0	No Account Privileges
193	0xC1	No Account Balance
194	0xC2	Credit Limit Exceeded
195	0xC3	Too Many Holds
196	0xC4	Account Disabled
232	0xE8	Write To Group
234	0xEA	No Such Member



Decimal	Hex	Description
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The amount, along with the requesting server's object ID, is recorded in the object's ACCOUNT\_HOLDS property. If the requesting server has an outstanding hold, the new hold amount is added to that server's outstanding hold amount.

You can back out a hold (in the event that the service is not provided) by calling **Submit Account Hold** with the additive inverse of the original amount held. You can also choose to partially back out a hold by submitting a smaller negative amount. To clear all holds, submit a hold with the amount set to zero.

If a service is provided, the hold should be cleared by passing the original hold amount plus the charge amount to **Submit Account Charge**.

*SubFuncStrucLen* = 8 + *ClientNameLen*

If the requesting server's object ID is not listed in the ACCOUNT\_SERVERS property of the file server's object, No Account Privileges (192) is returned.

If the specified object does not have an ACCOUNT\_BALANCE property, No Account Balance (193) is returned.

If the current balance minus all holds (including the requested hold) is less than the permissible amount, **Submit Account Hold** fails and Credit Limit Exceeded (194) is returned.

If too many other servers (16) have outstanding holds on the user's account, **Submit Account Hold** fails and Too Many Holds (195) is returned.

# Submit Account Note 0x2222 23 153

Allows a server to record a note about a client's account activities in the audit file.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (153)	byte
10	ServiceType	word (Hi-Lo)
12	ClientType	word (Hi-Lo)
14	CommentType	word (Hi-Lo)
16	ClientNameLen	byte
17	ClientName	byte[ClientNameLen]
17 + ClientNameLen	CommentLen	byte
18 + ClientNameLen	Comment	byte[CommentLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
150	0x96	Server Out Of Memory
192	0xC0	No Account Privileges
193	0xC1	No Account Balance
196	0xC4	Account Disabled
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set

Decimal	Hex	Description
240	0xF0	Illegal Wildcard
252	0xFC	No Such Object
255	0xFF	Hard Failure

## Remarks

**Submit Account Note** can be used by the file server, for example, to record a login time.

*SubFuncStrucLen* = 9 + *ClientNameLen* + *CommentLen*

If the requesting server is not listed in the ACCOUNT\_SERVERS property of the file server's object, No Account Privileges (192) is returned and no audit record is generated.





## AFP

AFP NCPs enable you to create, access, and delete Macintosh-format directories and files on a NetWare file server. NetWare support for AFP begins with NetWare 286 v2.15.

To access AFP NCPs, use the following:

- ♦ [Chapter 5, “Concepts,” on page 51](#)
- ♦ [Chapter 6, “NCPs,” on page 53](#)



# 5

## Concepts

This section explains ideas that are common to AFP NCPs.

### AFP File Names

AFP directory and file names (also called long names) are from 1-31 characters in length. A long name is preceded by one byte, which specifies the length of the name. Long names can contain any ASCII character between 1 and 255 (except a colon ':'), but they cannot be terminated by a NULL character (character zero).

The file server automatically generates DOS-style file names (short names) for all AFP directories and any files that are created or accessed and stores both the long name and short name for each AFP directory and file.

### AFP to DOS Conversions

NetWare converts AFP names to DOS names by the following specific conventions.

If a long name that contains no periods is converted to a short name, the first eight valid DOS characters of the long name are used:

Long Name	Short Name
THIS IS A NAME	THISISAN

If a long name contains a period within the first nine valid DOS characters, the first eight characters before the period and the first three characters after the period (up to, but not including, another period) are used as follows:

Long Name	Short Name
THIS.IS.A.NAME	THIS.IS

This guideline permits both AFP and non-AFP workstations to access AFP directories and files. For example, assume your application creates the following two files in a parent directory:

Long Name	Short Name
THIS IS THE FIRST FILE	THISISTH
THIS IS THE SECOND FILE	???

Since the two short names would be the same (THISISTH), the file server replaces the last character of the second file's short name with an ascending decimal number that will guarantee its uniqueness so that the short name of the second file would be THISIST1.

Another example follows:

Long Name	Short Name
THIS IS A 1 TIME OFFER	THISISA1
THIS IS A 1 TIME DEAL	THISISA2

Assume a Macintosh workstation copies THIS IS THE SECOND FILE (THISIST1) to a floppy disk and then copies the file from the disk to a different, empty directory on the file server. Since THIS IS THE SECOND FILE is the only file that would yield the short name THISISTH, the file server uses THISISTH as the short name.

## AFP Path Names

AFP paths use the NULL character (decimal zero) to distinguish between directory and file names as follows:

```
volume:directory1 directory2 file
```

This same path is represented on a NetWare file server as:

```
volume:director/director/file
```

NetWare applications frequently target the short name of a directory or a file by combining a NetWare directory handle and a short (NetWare style) directory or file path. Likewise, AFP applications target the long name of a directory or a file by combining an AFP entry ID and a long (AFP-style) directory or file path. You should not combine NetWare directory handles and long names, or AFP entry IDs and short names.



# 6

## NCPs

This section describes each of the AFP NCPs, their Request and Reply formats, and Return Values.

# AFP 2.0 Create Directory 0x2222 35 13

Creates a directory with an AFP directory name.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (46 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (13)	byte
10	VolumeNumber	byte
11	BaseDirectoryID	long (Hi-Lo)
15	Reserved	byte
16	FinderInfo	byte[32]
48	ProDOSInfo	byte[6]
54	PathLen	byte
55	PathModString	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
132	0x84	No Create Privileges
136	0x88	Invalid File Handle
147	0x93	No Read Privileges

Decimal	Hex	Description
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

*FinderInfo* is a 32-byte structure that is defined in Apple documentation.

*ProDOSInfo* is a 6-byte structure that is defined in Apple documentation.

## See Also

**AFP Delete 0x2222 35 03 (page 77), AFP Create Directory 0x2222 35 01 (page 73)**

# AFP 2.0 Create File 0x2222 35 14

Creates a file with an AFP file name.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (46 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (14)	byte
10	VolumeNumber	byte
11	BaseDirectoryID	long (Hi-Lo)
15	DeleteExistingFileFlag (0 = no, 1 = yes)	byte
16	FinderInfo	byte[32]
48	ProDOSInfo	byte[6]
54	PathLen	byte
55	PathModString	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
131	0x83	Hard I/O Error

Decimal	Hex	Description
132	0x84	No Create Privileges
135	0x87	Create File Name Error
136	0x88	Invalid File Handle
138	0x8A	No Delete Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

**AFP 2.0 Create File** does not open the resulting file. The file is created as a normal read/write file, and its system and hidden bits are cleared.

*PathModString* is in AFP long-name format.

*FinderInfo* is a 32-byte structure that is defined in Apple documentation.

*ProDOSInfo* is a 6-byte structure that is defined in Apple documentation.

## See Also

**AFP Delete 0x2222 35 03 (page 77), AFP Create File 0x2222 35 02 (page 75)**

# AFP 2.0 Get File Or Directory Information 0x2222 35 15

Returns the specified information for a file or directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (9 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (15)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	RequestBitMap	word (Hi-Lo)
17	PathStringLen	byte
18	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	AFPEntryID	long (Hi-Lo)
12	ParentID	long (Hi-Lo)
16	Attributes	word (Hi-Lo)
18	DataForkLen	long (Hi-Lo)
22	ResourceForkLen	long (Hi-Lo)
26	TotalOffspring	word (Hi-Lo)
28	CreationDate	word (Hi-Lo)
30	AccessDate	word (Hi-Lo)
32	ModifyDate	word (Hi-Lo)
34	ModifyTime	word (Hi-Lo)
36	BackupDate	word (Hi-Lo)

Offset	Content	Type
38	BackupTime	word (Hi-Lo)
40	FinderInfo	byte[32]
72	LongName	byte[32]
104	OwnerID	long (Hi-Lo)
108	ShortName	byte[12]
112	AccessPrivileges	word (Hi-Lo)
114	ProDOSInfo	byte[6]

## Parameters

### *AFPEntryID*

(Reply) Specifies the Apple equivalent to a 1-byte NetWare directory handle. A NetWare directory handle points to a file server volume or directory; an AFP entry ID points to a file server volume, directory, or file.

### *ParentID*

(Reply) Specifies the AFP entry ID for the parent directory of the target file or directory.

### *Attributes*

(Reply) Specifies the attributes of the directory or file as follows:

0x0001 Search Mode  
0x0002 Search Mode  
0x0004 Search Mode  
0x0008 Undefined  
0x0010 Transaction  
0x0020 Index  
0x0040 Read Audit  
0x0080 Write Audit  
0x0100 Read Only  
0x0200 Hidden  
0x0400 System  
0x0800 Execute Only  
0x1000 Subdirectory  
0x2000 Archive  
0x4000 Undefined  
0x8000 Shareable File

### *DataForkLen*

(Reply) Specifies the data size of the target AFP file. If *PathModString* specifies an AFP directory, *DataForkLen* is zero.

### *ResourceForkLen*

(Reply) Specifies the resource fork size of the target AFP file. If *PathModString* specifies an AFP directory, *ResourceForkLen* is zero.

### *TotalOffspring*

(Reply) Specifies the number of files and subdirectories contained within the specified directory. If the AFP directory or file path specifies an AFP file, *TotalOffspring* is zero.

### *CreationDate*

(Reply) Specifies the creation date (in AFP format) of the target directory or file.

### *AccessDate*

(Reply) Specifies when (in AFP format) the target AFP file was last accessed. If *PathModString* specifies an AFP directory, *AccessDate* is zero.

### *ModifyDate* and *ModifyTime*

(Reply) Specifies the last modified date and time (in AFP format) of the target AFP file. If *PathModString* specifies an AFP directory, these parameters are zero.

### *BackupDate* and *BackupTime*

(Reply) Specifies the last backup date and time (in AFP format) of the specified directory or file.

### *FinderInfo*

(Reply) Specifies the 32-byte finder information structure associated with each AFP directory or file.

### *LongName*

(Reply) Specifies the AFP directory or file name of the specified directory or file (1-31 characters).

### *OwnerID*

(Reply) Specifies the 4-byte binder object ID of the entity that created or last modified the file.

### *ShortName*

(Reply) Specifies the NetWare directory or file name of the specified directory or file (in DOS 8.3 format).

### *AccessPrivileges*

(Reply) Specifies a 1-word bit mask of the calling station's privileges for accessing the specified file or directory as follows:

0x0100 Read Privileges (files only)

0x0200 Write Privileges (files only)

0x0400 Open Privileges (files only)

0x0800 Create Privileges (files only)

0x1000 Delete Privileges (files only)

0x2000 Parental Privileges (directories only for creating, deleting, and renaming subdirectories)

0x4000 Search Privileges (directories only)

0x8000 Modify File Status Flags Privileges (files and directories)



(Reply) Specifies a 6-byte structure that is defined in Apple documentation.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

The *AFPEntryID* of the root is always 1L. The *ParentID* of the root is always 0L.

The returned information is determined by the bits that are set in *RequestBitMap* as follows:

0x0001 AFP Entry ID  
 0x0002 Data Fork Length  
 0x0004 Resource Fork Length  
 0x0008 Number of Offspring  
 0x0010 Owner Object ID  
 0x0020 Short Name  
 0x0040 Access Privileges  
 0x0100 Attributes  
 0x0200 AFP Parent Entry ID  
 0x0400 Creation Date  
 0x0800 Access Date  
 0x1000 Modify Date/Time  
 0x2000 Backup Date/Time  
 0x4000 Finder Information  
 0x8000 Long Name

If the returned object is a file, the bits in *AccessPrivileges* specify which rights the caller has in relation to the file and the rights that the caller has in the file's parent directory (which are

appropriately modified if the file itself has certain rights restrictions). For example, if the file rights are set to read only, the write and delete bits will be cleared.

If the returned object is a directory, the bits in *AccessPrivileges* specify which rights the caller has in that subdirectory.

## See Also

**AFP 2.0 Set File Information 0x2222 35 16 (page 68), AFP Get File Information 0x2222 35 05 (page 86), AFP Set File Information 0x2222 35 09 (page 101)**

# AFP 2.0 Scan File Information 0x2222 35 17

Returns information about an AFP entry (directory or file).

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (17 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (17)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	MacLastSeenID	long (Hi-Lo)
19	DesiredResponseCount	word (Hi-Lo)
21	SearchBitMap	word (Hi-Lo)
23	RequestBitMap	word (Hi-Lo)
25	PathStringLen	byte
26	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ActualResponseCount (repeats this number of times)	word (Hi-Lo)
10	EntryID	long (Hi-Lo)
14	ParentID	long (Hi-Lo)
18	Attributes	word (Hi-Lo)
20	DataForkLen	long (Hi-Lo)
24	ResourceForkLen	long (Hi-Lo)
28	TotalOffspring	word (Hi-Lo)

Offset	Content	Type
30	CreationDate	word (Hi-Lo)
32	AccessDate	word (Hi-Lo)
34	ModifyDate	word (Hi-Lo)
36	ModifyTime	word (Hi-Lo)
38	BackupDate	word (Hi-Lo)
40	BackupTime	word (Hi-Lo)
42	FinderInfo	byte[32]
74	LongName	byte[32]
106	OwnerID	long (Hi-Lo)
110	ShortName	byte[12]
122	AccessPrivileges	word (Hi-Lo)
124	ProDOSInfo	byte[6]

## Parameters

### *AFPEntryID*

(Reply) Specifies the Apple equivalent to a 1-byte NetWare directory handle. A NetWare directory handle points to a file server volume or directory; an AFP entry ID points to a file server volume, directory, or file.

### *ParentID*

(Reply) Specifies the AFP entry ID for the parent directory of the target file or directory.

### *Attributes*

(Reply) Specifies the attributes of the directory or file as follows:

0x0001 Search Mode  
 0x0002 Search Mode  
 0x0004 Search Mode  
 0x0008 Undefined  
 0x0010 Transaction  
 0x0020 Index  
 0x0040 Read Audit  
 0x0080 Write Audit  
 0x0100 Read Only  
 0x0200 Hidden  
 0x0400 System  
 0x0800 Execute Only  
 0x1000 Subdirectory  
 0x2000 Archive

0x4000 Undefined  
0x8000 Shareable File

*DataForkLen*

(Reply) Specifies the data size of the target AFP file. If *PathModString* specifies an AFP directory, *DataForkLen* is zero.

*ResourceForkLen*

(Reply) Specifies the resource fork size of the target AFP file. If *PathModString* specifies an AFP directory, *ResourceForkLen* is zero.

*TotalOffspring*

(Reply) Specifies the number of files and subdirectories contained within the specified directory. If the AFP directory or file path specifies an AFP file, *TotalOffspring* is zero.

*CreationDate*

(Reply) Specifies the creation date (in AFP format) of the target directory or file.

*AccessDate*

(Reply) Specifies when (in AFP format) the target AFP file was last accessed. If *PathModString* specifies an AFP directory, *AccessDate* is zero.

*ModifyDate* and *ModifyTime*

(Reply) Specifies the last modified date and time (in AFP format) of the target AFP file. If *PathModString* specifies an AFP directory, these parameters are zero.

*BackupDate* and *BackupTime*

(Reply) Specifies the last backup date and time (in AFP format) of the specified directory or file.

*FinderInfo*

(Reply) Specifies the 32-byte finder information structure associated with each AFP directory or file.

*LongName*

(Reply) Specifies the AFP directory or file name of the specified directory or file (1-31 characters).

*OwnerID*

(Reply) Specifies the 4-byte binder object ID of the entity that created or last modified the file.

*ShortName*

(Reply) Specifies the NetWare directory or file name of the specified directory or file (in DOS 8.3 format).

*AccessPrivileges*

(Reply) Specifies a 1-word bit mask of the calling station's privileges for accessing the specified file or directory as follows:

0x0100 Read Privileges (files only)  
0x0200 Write Privileges (files only)  
0x0400 Open Privileges (files only)

0x0800 Create Privileges (files only)  
 0x1000 Delete Privileges (files only)  
 0x2000 Parental Privileges (directories only for creating, deleting, and renaming subdirectories)  
 0x4000 Search Privileges (directories only)  
 0x8000 Modify File Status Flags Privileges (files and directories)

#### *ProDOSInfo*

(Reply) Specifies a 6-byte structure that is defined in Apple documentation.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

**AFP 2.0 Scan File Information** supports iterative directory scanning. You can specify up to four responses per request in *DesiredResponseCount*.

To scan all files and subdirectories of a directory, set *MacLastSeenID* to -1L on the first request; on each subsequent request, set *MacLastSeenID* to the *EntryID* of the previous request.

If *PathModString* specifies a directory, all entries within the directory that fit the *SearchBitMap* and *RequestBitMap* criteria will be returned.

*SearchBitMap* can have the following values:

0x0100 Search For Hidden Files/Directories  
 0x0200 Search For System Files/Directories  
 0x0400 Search For Subdirectories  
 0x0800 Search For Files

*RequestBitMap* can have the following values:

0x0001	Entry ID
0x0002	Data Fork Length
0x0004	Resource Fork Length
0x0008	Number of Offspring
0x0010	Owner ID
0x0020	Short Name
0x0040	Access Rights
0x0100	Attributes
0x0200	Parent Directory ID
0x0400	Creation Date
0x0800	Access Date
0x1000	Modify Date/Time
0x2000	Backup Date/Time
0x4000	Finder Information
0x8000	Long Name

If the returned object is a file, the bits in *AccessPrivileges* specify which rights the caller has in relation to the file and the rights that the caller has in the file's parent directory (which are appropriately modified if the file itself has certain rights restrictions). For example, if the file rights are set to read only, the write and delete bits will be cleared.

If the returned object is a directory, the bits in *AccessPrivileges* specify which rights the caller has in that subdirectory.

## See Also

**AFP 2.0 Set File Information 0x2222 35 16 (page 68), AFP Scan File Information 0x2222 35 10 (page 96)**

# AFP 2.0 Set File Information 0x2222 35 16

Sets information pertaining to the specified AFP file or directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (61 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (16)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	RequestBitMap	word (Hi-Lo)
17	Attributes	word (Hi-Lo)
19	CreationDate	word (Hi-Lo)
21	AccessDate	word (Hi-Lo)
23	ModifyDate	word (Hi-Lo)
25	ModifyTime	word (Hi-Lo)
27	BackupDate	word (Hi-Lo)
29	BackupTime	word (Hi-Lo)
31	FinderInfo	byte[32]
63	ProDOSInfo	byte[6]
69	PathStringLen	byte
70	PathModString	byte[PathStringLen]

## Parameters

### *Attributes*

(Request) Specifies the attributes of the directory or file as follows:

- 0x0001 Search Mode
- 0x0002 Search Mode
- 0x0004 Search Mode
- 0x0008 Undefined



0x0010 Transaction  
 0x0020 Index  
 0x0040 Read Audit  
 0x0080 Write Audit  
 0x0100 Read Only  
 0x0200 Hidden  
 0x0400 System  
 0x0800 Execute Only  
 0x1000 Subdirectory  
 0x2000 Archive  
 0x4000 Undefined  
 0x8000 Shareable File

#### *CreationDate*

(Request) Specifies the creation date (in AFP format) of the target directory or file.

#### *AccessDate*

(Request) Specifies when (in AFP format) the target AFP file was last accessed (ignored for directories).

#### *ModifyDate* and *ModifyTime*

(Request) Specifies the last modified date and time (in AFP format) of the target AFP file (ignored for directories).

#### *BackupDate* and *BackupTime*

(Request) Specifies the last backup date and time (in AFP format) of the specified directory or file.

#### *FinderInfo*

(Request) Specifies the 32-byte finder information structure associated with the specified AFP directory or file.

#### *ProDOSInfo*

(Request) Specifies a 6-byte structure that is defined in Apple documentation.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
148	0x94	No Write Privileges

Decimal	Hex	Description
149	0x95	File Detached
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

*RequestBitMap* can have the following values:

0x0100 Attributes  
0x0400 Creation Date  
0x0800 Access Date  
0x1000 Modify Date/Time  
0x2000 Backup Date/Time  
0x4000 Finder Information

## See Also

**AFP Get File Information 0x2222 35 05 (page 86), AFP 2.0 Scan File Information 0x2222 35 17 (page 63)**

# AFP Alloc Temporary Directory Handle 0x2222 35 11

Maps a NetWare directory handle to an AFP directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (7 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (11)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	PathStringLen	byte
16	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NetWareDirectoryHandle	byte
9	NetWareAccessRights	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error

Decimal	Hex	Description
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
157	0x9D	No Directory Handles
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

*NetWareAccessRights* is a 1-byte mask that returns the effective rights that the calling station has in the target directory as follows:

0x01 Read  
0x02 Write  
0x04 Open  
0x08 Create  
0x10 Delete  
0x20 Parental  
0x40 Search  
0x80 Modify

# AFP Create Directory 0x2222 35 01

Creates a directory with an AFP directory name.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (40 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (1)	byte
10	VolumeNumber	byte
11	BaseDirectoryID	long (Hi-Lo)
15	Reserved	byte
16	FinderInfo	byte[32]
48	PathLen	byte
49	PathModString	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
132	0x84	No Create Privileges
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory

Decimal	Hex	Description
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

*FinderInfo* is a 32-byte structure that is defined in Apple documentation.

## See Also

**AFP Delete 0x2222 35 03 (page 77), AFP 2.0 Create Directory 0x2222 35 13 (page 54)**

# AFP Create File 0x2222 35 02

Creates a file with an AFP file name.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (40 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (2)	byte
10	VolumeNumber	byte
11	BaseDirectoryID	long (Hi-Lo)
15	DeleteExistingFileFlag (0 = no, 1 = yes)	byte
16	FinderInfo	byte[32]
48	PathLen	byte
49	PathModString	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
131	0x83	Hard I/O Error
132	0x84	No Create Privileges

Decimal	Hex	Description
135	0x87	Create File Name Error
136	0x88	Invalid File Handle
138	0x8A	No Delete Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

**AFP Create File** does not open the resulting file. The file is created as a normal read/write file, and its system and hidden bits are cleared.

*PathModString* is in AFP long-name format.

## See Also

**AFP Delete 0x2222 35 03 (page 77), AFP 2.0 Create File 0x2222 35 14 (page 56)**



# AFP Delete 0x2222 35 03

Deleted a file or directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (7 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (3)	byte
10	VolumeNumber	byte
11	BaseDirectoryID	long (Hi-Lo)
15	PathLen	byte
16	PathModString	byte[PathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
138	0x8A	No Delete Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path

Decimal	Hex	Description
158	0x9E	Bad File Name
159	0x9F	Directory Active
160	0xA0	Directory Not Empty
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

Directories that are to be deleted must be empty. Files that are to be deleted must be closed by all users.

*PathModString* can be NULL.

By itself, *BaseDirectoryID* can specify the file or directory to be deleted.

## See Also

**AFP Create File 0x2222 35 02 (page 75), AFP 2.0 Create File 0x2222 35 14 (page 56), AFP 2.0 Create Directory 0x2222 35 13 (page 54), AFP Create Directory 0x2222 35 01 (page 73), AFP Get Macintosh Info On Deleted File 0x2222 35 19 (page 91)**

# AFP Get DOS Name From Entry ID 0x2222 35 18

Returns the DOS directory path that corresponds to a 32-bit Macintosh file or directory entry ID.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (18)	byte
10	VolumeNumber	byte
11	MacDirectoryEntryID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	PathStringLen	byte
9	DOSPathString	byte[PathStringLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
137	0x89	No Search Privileges
150	0x96	Server Out of Memory
191	0xBF	Invalid Name Space

## Remarks

*MacDirectoryEntryID* is the unique 32-bit AFP identifier for the file or directory.

## See Also

[AFP Get Entry ID From Path Name 0x2222 35 12 \(page 84\)](#), [AFP Get Entry ID From Name 0x2222 35 04 \(page 81\)](#)

# AFP Get Entry ID From Name 0x2222 35 04

Returns the 32-bit AFP ID number for the specified directory or file (long name).

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (7 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (4)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	PathStringLen	byte
16	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	TargetEntryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path

Decimal	Hex	Description
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

The ID of the root directory is 1L.

## See Also

**AFP Get DOS Name From Entry ID 0x2222 35 18 (page 79)**

# AFP Get Entry ID From NetWare Handle 0x2222 35 06

Returns an AFP Entry ID from the specified NetWare file handle.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (6)	byte
10	NetWareFileHandle	byte[6]

## Reply Format

Offset	Content	Type
Reply header		
8	VolumelD	byte
9	TargetEntryID	long (Hi-Lo)
13	ForkIndicator (0 = data, 1 = resource)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
162	0xA2	I/O Lock Error

# AFP Get Entry ID From Path Name 0x2222 35 12

Converts a NetWare style path specification into a unique 32-bit Macintosh file or directory entry ID.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (3 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (12)	byte
10	NetWareDirectoryHandle	byte
11	PathStringLen	byte
12	NetWarePathString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	TargetEntryID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path



Decimal	Hex	Description
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

*NetWareDirectoryHandle* and *NetWarePathString* are given in NetWare format (short name).

*TargetEntryID* is the unique 32-bit AFP identifier for the file or directory.

## See Also

**AFP Get DOS Name From Entry ID 0x2222 35 18 (page 79)**

# AFP Get File Information 0x2222 35 05

Returns the specified information for a file.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (9 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (5)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	RequestBitMap	word (Hi-Lo)
17	PathStringLen	byte
18	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	AFPEntryID	long (Hi-Lo)
12	ParentID	long (Hi-Lo)
16	Attributes	word (Hi-Lo)
18	DataForkLen	long (Hi-Lo)
22	ResourceForkLen	long (Hi-Lo)
26	TotalOffspring	word (Hi-Lo)
28	CreationDate	word (Hi-Lo)
30	AccessDate	word (Hi-Lo)
32	ModifyDate	word (Hi-Lo)
34	ModifyTime	word (Hi-Lo)
36	BackupDate	word (Hi-Lo)

Offset	Content	Type
38	BackupTime	word (Hi-Lo)
40	FinderInfo	byte[32]
72	LongName	byte[32]
104	OwnerID	long (Hi-Lo)
108	ShortName	byte[12]
112	AccessPrivileges	word (Hi-Lo)

## Parameters

### *AFPEntryID*

(Reply) Specifies the Apple equivalent to a 1-byte NetWare directory handle. A NetWare directory handle points to a file server volume or directory; an AFP entry ID points to a file server volume, directory, or file.

### *ParentID*

(Reply) Specifies the AFP entry ID for the parent directory of the target file.

### *Attributes*

(Reply) Specifies the attributes of the directory or file as follows:

0x0001 Search Mode  
0x0002 Search Mode  
0x0004 Search Mode  
0x0008 Undefined  
0x0010 Transaction  
0x0020 Index  
0x0040 Read Audit  
0x0080 Write Audit  
0x0100 Read Only  
0x0200 Hidden  
0x0400 System  
0x0800 Execute Only  
0x1000 Subdirectory  
0x2000 Archive  
0x4000 Undefined  
0x8000 Shareable File

### *DataForkLen*

(Reply) Specifies the data size of the target AFP file. If *PathModString* specifies an AFP directory, *DataForkLen* is zero.

### *ResourceForkLen*

(Reply) Specifies the resource fork size of the target AFP file. If *PathModString* specifies an AFP directory, *ResourceForkLen* is zero.

### *TotalOffspring*

(Reply) Specifies the number of files and subdirectories contained within the specified directory. If the AFP directory or file path specifies an AFP file, *TotalOffspring* is zero.

### *CreationDate*

(Reply) Specifies the creation date (in AFP format) of the target directory or file.

### *AccessDate*

(Reply) Specifies when (in AFP format) the target AFP file was last accessed. If *PathModString* specifies an AFP directory, *AccessDate* is zero.

### *ModifyDate* and *ModifyTime*

(Reply) Specifies the last modified date and time (in AFP format) of the target AFP file. If *PathModString* specifies an AFP directory, these parameters are zero.

### *BackupDate* and *BackupTime*

(Reply) Specifies the last backup date and time (in AFP format) of the specified directory or file.

### *FinderInfo*

(Reply) Specifies the 32-byte finder information structure associated with each AFP directory or file.

### *LongName*

(Reply) Specifies the AFP directory or file name of the specified directory or file (1-31 characters).

### *OwnerID*

(Reply) Specifies the 4-byte binder object ID of the entity that created or last modified the file.

### *ShortName*

(Reply) Specifies the NetWare directory or file name of the specified directory or file (in DOS 8.3 format).

### *AccessPrivileges*

(Reply) Specifies a 1-word bit mask of the calling station's privileges for accessing the specified file or directory as follows:

- 0x0100 Read Privileges (files only)
- 0x0200 Write Privileges (files only)
- 0x0400 Open Privileges (files only)
- 0x0800 Create Privileges (files only)
- 0x1000 Delete Privileges (files only)
- 0x2000 Parental Privileges (directories only for creating, deleting, and renaming subdirectories)
- 0x4000 Search Privileges (directories only)
- 0x8000 Modify File Status Flags Privileges (files and directories)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

The *AFPEntryID* of the root is always 1L. The *ParentID* of the root is always 0L.

The returned information is determined by the bits that are set in *RequestBitMap* as follows:

0x0001 AFP Entry ID  
0x0002 Data Fork Length  
0x0004 Resource Fork Length  
0x0008 Number of Offspring  
0x0010 Owner Object ID  
0x0020 Short Name  
0x0040 Access Privileges  
0x0100 Attributes  
0x0200 AFP Parent Entry ID  
0x0400 Creation Date  
0x0800 Access Date  
0x1000 Modify Date/Time  
0x2000 Backup Date/Time  
0x4000 Finder Information  
0x8000 Long Name

If the returned object is a file, the bits in *AccessPrivileges* specify which rights the caller has in relation to the file and the rights that the caller has in the file's parent directory (which are appropriately modified if the file itself has certain rights restrictions). For example, if the file rights are set to read only, the write and delete bits will be cleared.

## See Also

[AFP 2.0 Set File Information 0x2222 35 16 \(page 68\)](#), [AFP Set File Information 0x2222 35 09 \(page 101\)](#)

# AFP Get Macintosh Info On Deleted File 0x2222 35 19

Returns the Finder and ProDOS structures for a deleted Macintosh directory entry.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (19)	byte
10	VolumeNumber	byte
11	DOSDirectoryNumber	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	FinderInfo	byte[32]
40	ProDOSInfo	byte[6]
46	ResourceForkSize	long (Hi-Lo)
50	FileNameLen	byte
51	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
156	0x9C	Invalid Path
191	0xBF	Invalid Name Space

## See Also

**AFP Delete 0x2222 35 03 (page 77)**

# AFP Open File Fork 0x2222 35 08

Opens an AFP file fork (either a data fork or a resource fork).

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (9 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (8)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	ForkIndicator (0 = data, 1 = resource)	byte
16	AccessMode	byte
17	PathStringLen	byte
18	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	EntryID	long (Hi-Lo)
12	ForkLen	long (Hi-Lo)
16	NetWareAccessHandle	byte[6]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles



Decimal	Hex	Description
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
148	0x94	No Write Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

If a nonexistent file fork is specified, **AFP Open File Fork** automatically creates and opens a file fork.

*AccessMode* can have the following values:

- 0x01 Read Access
- 0x02 Write Access
- 0x04 Deny Read Access
- 0x08 Deny Write Access
- 0x10 Compatibility Mode Bit (should always be set)

# AFP Rename 0x2222 35 07

Moves or renames an AFP file or directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (7)	byte
10	VolumeNumber	byte
11	MacSourceBaseID	long (Hi-Lo)
15	MacDestinationBaseID	long (Hi-Lo)
19	SourcePathLen	byte
20	SourcePathString	byte[SourcePathLen]
20 + SourcePathLen	DestinationPathLen	byte
21 + SourcePathLen	DestinationPathString	byte[DestinationPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
132	0x84	No Create Privileges
136	0x88	Invalid File Handle
139	0x8B	No Rename Privileges
142	0x8E	All Files In Use
144	0x90	All Read Only
146	0x92	All Names Exist
147	0x93	No Read Privileges
150	0x96	Server Out of Memory

Decimal	Hex	Description
152	0x98	Disk Map Error
153	0x99	Directory Full Error
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

*SourcePathString* must be set to the old directory or file name, and *DestinationPathString* must be set to the new name.

If the file or directory is being renamed but not moved, *DestinationBaseID* should be set to the file's current parent directory; and *DestinationPathString* should be set to the new name of the file or directory.

If the file or directory is being moved but not renamed, *DestinationBaseID* should be set to the ID of the target directory; and *DestinationPathLen* should be set to zero.

If the file or directory is being moved and renamed, the last item in *DestinationPathString* is used as the new name of the file or directory.

$SubFuncStructLen = 12 + SourcePathLen + DestinationPathLen$

# AFP Scan File Information 0x2222 35 10

Returns information about an AFP entry (directory or file).

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (17 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (10)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	MacLastSeenID	long (Hi-Lo)
19	DesiredResponseCount	word (Hi-Lo)
21	SearchBitMap	word (Hi-Lo)
23	RequestBitMap	word (Hi-Lo)
25	PathStringLen	byte
26	PathModString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ActualResponseCount (repeats this number of times)	word (Hi-Lo)
10	EntryID	long (Hi-Lo)
14	ParentID	long (Hi-Lo)
18	Attributes	word (Hi-Lo)
20	DataForkLen	long (Hi-Lo)
24	ResourceForkLen	long (Hi-Lo)
28	TotalOffspring	word (Hi-Lo)

Offset	Content	Type
30	CreationDate	word (Hi-Lo)
32	AccessDate	word (Hi-Lo)
34	ModifyDate	word (Hi-Lo)
36	ModifyTime	word (Hi-Lo)
38	BackupDate	word (Hi-Lo)
40	BackupTime	word (Hi-Lo)
42	FinderInfo	byte[32]
74	LongName	byte[32]
106	OwnerID	long (Hi-Lo)
110	ShortName	byte[12]
122	AccessPrivileges	word (Hi-Lo)

## Parameters

### *EntryID*

(Reply) Specifies the Apple equivalent to a 1-byte NetWare directory handle. A NetWare directory handle points to a file server volume or directory; an AFP entry ID points to a file server volume, directory, or file.

### *ParentID*

(Reply) Specifies the AFP entry ID for the parent directory of the target file.

### *Attributes*

(Reply) Specifies the attributes of the directory or file as follows:

0x0001 Search Mode  
 0x0002 Search Mode  
 0x0004 Search Mode  
 0x0008 Undefined  
 0x0010 Transaction  
 0x0020 Index  
 0x0040 Read Audit  
 0x0080 Write Audit  
 0x0100 Read Only  
 0x0200 Hidden  
 0x0400 System  
 0x0800 Execute Only  
 0x1000 Subdirectory  
 0x2000 Archive  
 0x4000 Undefined  
 0x8000 Shareable File

### *DataForkLen*

(Reply) Specifies the data size of the target AFP file. If *PathModString* specifies an AFP directory, *DataForkLen* is zero.

### *ResourceForkLen*

(Reply) Specifies the resource fork size of the target AFP file. If *PathModString* specifies an AFP directory, *ResourceForkLen* is zero.

### *TotalOffspring*

(Reply) Specifies the number of files and subdirectories contained within the specified directory. If the AFP directory or file path specifies an AFP file, *TotalOffspring* is zero.

### *CreationDate*

(Reply) Specifies the creation date (in AFP format) of the target directory or file.

### *AccessDate*

(Reply) Specifies when (in AFP format) the target AFP file was last accessed. If *PathModString* specifies an AFP directory, *AccessDate* is zero.

### *ModifyDate* and *ModifyTime*

(Reply) Specifies the last modified date and time (in AFP format) of the target AFP file. If *PathModString* specifies an AFP directory, these parameters are zero.

### *BackupDate* and *BackupTime*

(Reply) Specifies the last backup date and time (in AFP format) of the specified directory or file.

### *FinderInfo*

(Reply) Specifies the 32-byte finder information structure associated with each AFP directory or file.

### *LongName*

(Reply) Specifies the AFP directory or file name of the specified directory or file (1-31 characters).

### *OwnerID*

(Reply) Specifies the 4-byte binder object ID of the entity that created or last modified the file.

### *ShortName*

(Reply) Specifies the NetWare directory or file name of the specified directory or file (in DOS 8.3 format).

### *AccessPrivileges*

(Reply) Specifies a 1-word bit mask of the calling station's privileges for accessing the specified file or directory as follows:

- 0x0100 Read Privileges (files only)
- 0x0200 Write Privileges (files only)
- 0x0400 Open Privileges (files only)
- 0x0800 Create Privileges (files only)
- 0x1000 Delete Privileges (files only)

0x2000 Parental Privileges (directories only for creating, deleting, and renaming subdirectories)  
 0x4000 Search Privileges (directories only)  
 0x8000 Modify File Status Flags Privileges (files and directories)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

**AFP Scan File Information** supports iterative directory scanning; up to four responses per request can be specified in *DesiredResponseCount*.

To scan all files and subdirectories of a directory, set *MacLastSeenID* to -1L on the first request. For each subsequent request, set *MacLastSeenID* to the *EntryID* of the previous request.

If *PathModString* specifies a directory, all entries within the directory that fit the search criteria specified in *SearchBitMap* and *RequestBitMap* will be returned.

*SearchBitMap* can have the following values:

0x0100 Hidden Files/Directories  
 0x0200 System Files/Directories  
 0x0400 Subdirectories  
 0x0800 Files

*RequestBitMap* can have the following values:

0x0001 Entry ID  
 0x0002 Data Fork Length  
 0x0004 Resource Fork Length  
 0x0008 Number of Offspring

0x0010	Owner ID
0x0020	Short Name
0x0040	Access Privileges
0x0100	Attributes
0x0200	Parent Directory ID
0x0400	Creation Date
0x0800	Access Date
0x1000	Modify Date/Time
0x2000	Backup Date/Time
0x4000	Finder Information
0x8000	Long Name

If the returned object is a file, the bits in *AccessPrivileges* specify which rights the caller has in relation to the file and the rights that the caller has in the file's parent directory (which are appropriately modified if the file itself has certain rights restrictions). For example, if the file rights are set to read only, the write and delete bits will be cleared.

If the returned object is a directory, the bits in *AccessPrivileges* specify which rights the caller has in that directory.

## See Also

**AFP 2.0 Scan File Information 0x2222 35 17 (page 63), AFP Set File Information 0x2222 35 09 (page 101)**



# AFP Set File Information 0x2222 35 09

Sets information pertaining to the specified AFP file or directory.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (35)	byte
7	SubFuncStrucLen (55 + PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (09)	byte
10	VolumeNumber	byte
11	MacBaseDirectoryID	long (Hi-Lo)
15	RequestBitMap	word (Hi-Lo)
17	Attributes	word (Hi-Lo)
19	CreationDate	word (Hi-Lo)
21	AccessDate	word (Hi-Lo)
23	ModifyDate	word (Hi-Lo)
25	ModifyTime	word (Hi-Lo)
27	BackupDate	word (Hi-Lo)
29	BackupTime	word (Hi-Lo)
31	FinderInfo	byte[32]
63	PathStringLen	byte
64	PathModString	byte[PathStringLen]

## Parameters

### *Attributes*

(Request) Specifies the attributes of the directory or file as follows:

- 0x0001 Search Mode
- 0x0002 Search Mode
- 0x0004 Search Mode
- 0x0008 Undefined
- 0x0010 Transaction

0x0020 Index  
 0x0040 Read Audit  
 0x0080 Write Audit  
 0x0100 Read Only  
 0x0200 Hidden  
 0x0400 System  
 0x0800 Execute Only  
 0x1000 Subdirectory  
 0x2000 Archive  
 0x4000 Undefined  
 0x8000 Shareable File

#### *CreationDate*

(Request) Specifies the creation date (in AFP format) of the target directory or file.

#### *AccessDate*

(Request) Specifies when (in AFP format) the target AFP file was last accessed (ignored for directories).

#### *ModifyDate* and *ModifyTime*

(Request) Specifies the last modified date and time (in AFP format) of the target AFP file (ignored for directories).

#### *BackupDate* and *BackupTime*

(Request) Specifies the last backup date and time (in AFP format) of the specified directory or file.

#### *FinderInfo*

(Request) Specifies the 32-byte finder information structure associated with the specified AFP directory or file.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
148	0x94	No Write Privileges
149	0x95	File Detached
150	0x96	Server Out of Memory
152	0x98	Disk Map Error

Decimal	Hex	Description
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	Failure, File Exists Error

## Remarks

*RequestBitMap* can have the following values:

0x0100 Attributes  
0x0400 Creation Date  
0x0800 Access Date  
0x1000 Modify Date/Time  
0x2000 Backup Date/Time  
0x4000 Finder Information

## See Also

**AFP Scan File Information 0x2222 35 10 (page 96), AFP 2.0 Set File Information 0x2222 35 16 (page 68)**



# IV

## Bindery

Each NetWare file server includes a small database called a bindery. The bindery is used by the file server to maintain information about various resources on the local network and to help administer network security. Novell's implementation of the bindery uses hidden files that are located in the SYS:SYSTEM directory. The number of hidden files varies with the particular version of NetWare.

To access Bindery NCPs, use the following:

- ♦ [Chapter 7, "Concepts," on page 107](#)
- ♦ [Chapter 8, "NCPs," on page 115](#)
- ♦ [Chapter 9, "Structures," on page 173](#)



# 7

## Concepts

This section explains ideas that are common to Bindery NCPs.

### Bindery Objects

The NetWare operating system maintains a list (in the bindery files) of all objects allowed to access the file server. NetWare also records information about each object. A bindery object can be a user, a user group, a file server, a print server, or any other named entity that might gain access to a file server. Each bindery object record consists of the following information:

Offset	Content	Type
0	ObjectID: Uniquely identifies the object within a particular file server's bindery (4 bytes).	dword
4	<a href="#">"ObjectType" on page 107</a> : Specifies whether the object is a user, a user group, a file server, etc.	word
6	<a href="#">"ObjectName" on page 108</a> : Specifies the name of an object in a NULL-terminated string (1-47 characters).	byte[48]
54	ObjectFlag: Specifies whether the object is static (00h) or dynamic (01h). A static object exists in a bindery until an application explicitly deletes it with the Delete Bindery Object request. A dynamic object disappears from a file server's bindery when the file server is rebooted. (If an object is a service-advertising server, the object disappears from a bindery when the server stops advertising.)	byte
55	<a href="#">ObjectSecurity</a> : Specifies who can access the bindery. The low-order bit determines who can read (scan for and find) the object. The high-order bit determines who can write to (add properties to or delete properties from) an object.	byte
56	PropertiesFlag: Specifies whether one or more properties are associated with the object (00h = no properties, FFh = one or more properties).	byte

### ObjectType

Object types up to and including 8000h are reserved by Novell. If you want to define a well-known object, contact Novell. You can define object types above 8000h for your own development and application.

*ObjectType* can be one of the following values:

FFFFh (-1) WILD

0000h UNKNOWN

0001h USER  
 0002h USER\_GROUP  
 0003h PRINT\_QUEUE  
 0004h FILE\_SERVER  
 0005h JOB\_SERVER  
 0006h GATEWAY  
 0007h PRINT\_SERVER  
 0008h ARCHIVE\_QUEUE  
 0009h ARCHIVE\_SERVER  
 000Ah JOB\_QUEUE  
 000Bh ADMINISTRATION  
 0021h NAS\_SNA\_GATEWAY  
 0024h REMOTE\_BRIDGE\_SERVER  
 0047h ADVERTISING\_PRINT\_SERVER  
 0000h-8000h Reserved

## ObjectName

Only printable characters can be used; *ObjectName* cannot include spaces or the following characters:

/ slash  
 \ backslash  
 : colon  
 ; semicolon  
 , comma  
 \* asterisk  
 ? question mark

## ObjectSecurity

*ObjectSecurity* is defined as follows:

Value	Bits	Definition
0	0 0 0 0	Anyone: Access is allowed to all clients, even if the client has not logged in to the server with Login Object (0x2222 23 20).
1	0 0 0 1	Logged (in): Any client can access this object if the client has successfully logged in to the file server using Login Object (0x2222 23 20).
2	0 0 1 0	Object: A client can access this object only if the client has identified itself to the file server using the Login Object request that contained this object's name, type, and password.
3	0 0 1 1	Supervisor: A client can access this object only if the client has identified itself to the server with the Login Object request that contained the name and password of the object supervisor or an equivalent object.
4	0 1 0 0	File Server: Only the file server is allowed access to this object.



An object with *ObjectSecurity* set to 31h indicates that any user logged in to the file server can read or find the object. However, only the supervisor (or equivalent object) can write or add a property to the object. This object is also referred to as a "logged read, supervisor write" object.

## Properties and Values

Each bindery object might have one or more properties associated with it. For example, the object DAN (ObjectType 0001h) might have the GROUPS\_IM\_IN, ACCOUNT\_BALANCE, and PASSWORD properties associated with it. Properties only identify categories of information associated with the object.

**NOTE:** GROUPS\_IM\_IN is not the name of a user group to which the object belongs; it is only the name of one category of information associated with that object. Similarly, ACCOUNT\_BALANCE is not an actual numerical balance; and PASSWORD is not an actual password.

A property might have one or more values associated with it. A value associated with the GROUPS\_IM\_IN property must be the Object ID of a user group to which DAN belongs. The value of the ACCOUNT\_BALANCE property must be user DAN's current balance. The value of the PASSWORD property must be DAN's login password.

Properties fall into one of the following categories:

- ◆ Item properties
- ◆ Set properties

Item properties have values consisting of a 128-byte ASCII string. ACCOUNT\_BALANCE and PASSWORD are item properties. The value associated with ACCOUNT\_BALANCE is a monetary balance contained in the first few bytes of the 128-byte string (zeros fill the remaining bytes).

For NetWare 2.1 and above, PASSWORD contains an encrypted value. For NetWare 2.0 and previous versions, PASSWORD contains a 1-byte to 128-byte password (zeros fill the remaining bytes).

Set properties have values consisting of Object ID lists that are contained in one or more 128-byte segments, with each segment containing between 1-32 Object IDs. Each Object ID is a dword (4 bytes). GROUPS\_IM\_IN is a set property; its 128-byte segment contains the Object IDs of 1-32 user groups to which DAN belongs.

Each property record consists of the following information:

Offset	Content	Type
0	<u>PropertyName</u> : Specifies the name of the property (1-15 characters).	byte[15]
16	<u>PropertyFlags</u> : Specifies whether the property is static or dynamic and whether it is an item property or set property.	byte
17	<u>PropertySecurity</u> : Specifies who can access the property.	byte
18	ValuesFlag: Specifies whether the property has an associated value.	byte

## PropertyName

Only printable characters can be used; *PropertyName* cannot include spaces or the following characters:

- / slash
- \ backslash
- : colon
- ; semicolon
- , comma
- \* asterisk
- ? question mark

## PropertyFlags

Bit 0 is the static/dynamic flag, which is defined as follows:

7	6	5	4	3	2	1	0	Definition
							0	Static property
							1	Dynamic property

A static property exists until it is deleted with the **Delete Bindery Object 0x2222 23 51 (page 131)** request. A dynamic property is flushed from the file server's bindery when the file server is rebooted.

Bit 1 is the item/set flag, which is defined as follows:

7	6	5	4	3	2	1	0	Definition
						0		Item property
						1		Set property

The bindery process interprets the value of an item property as a 128-byte string and interprets the value of a set property as a series of Object ID numbers, each of which is 4 bytes long.

For example, the following bit combination indicates a static, set property:

7	6	5	4	3	2	1	0
						1	0
						1	

## PropertySecurity

The low-order bit determines who can read (scan for and find) the property; the high-order bit determines who can write (add values) to the property as follows:

Value	Bits	Definition
0	0 0 0 0	Anyone: Access is allowed to all clients, even if the client has not logged in to the server with Login Object (0x2222 23 20).
1	0 0 0 1	Logged (in): Any client can access this object if the client has successfully logged in to the file server using Login Object (0x2222 23 20).
2	0 0 1 0	Object: A client can access this object only if the client has identified itself to the file server using the Login Object request that contained this object's name, type, and password.
3	0 0 1 1	Supervisor: A client can access this object only if the client has identified itself to the server with the Login Object request that contained the name and password of the object supervisor or an equivalent object.
4	0 1 0 0	File Server: Only the file server is allowed access to this object.

For example, 31h indicates that any user logged in to the file server can find and read the property, but only the supervisor can add values to or change the property. This property is referred to as a "logged read, supervisor write" property.

## Property List

The following list contains well-known properties and the property type and object type that can be associated with each property:

Property Name	Type	Associated Objects
ACCOUNT_SERVERS	Set	Any object of type FILE_SERVER
ACCOUNT_BALANCE	Item	Any object that can log in and be charged
GROUP_MEMBERS	Set	Objects (users) that are members of some named group (a property that is usually attached to an object of type USER_GROUP)
GROUPS_IM_IN	Set	Objects (groups) with a set GROUP_MEMBERS property where this object's ID appears (a property that is usually attached to an object of type USER)
IDENTIFICATION	Item	Any object (a property that is informational and is usually attached to object of type USER or USER_GROUP)
LOGIN_CONTROL	Item	Any object that can log in
MANAGERS	Set	The SUPERVISOR object
NET_ADDRESS	Item	An object of type FILE_SERVER or any other "advertised" object
OPERATOR	Set	Any object
PASSWORD	Item	Any object that can log in
SECURITY_EQUALS	Set	Any object that can log in

A property or object is visible only if you have read privileges to the item. Otherwise, the item will not be revealed when you scan the bindery.

## Workgroup Managers

NetWare 2.1 and above supports the workgroup manager concept. Many large organizations need workgroup managers because several groups share a single file server, but each group wants autonomous control over its own users and data.

For example, a writing department and an editing department may want to use the same file server, but each may also want to have control over its own users and data. The function of a workgroup manager is similar to that of the network supervisor with certain limitations. The workgroup manager can create and delete users and groups and administer data resources, but only for part of the user database (bindery) and only for part of the logical file system.

A new set property, called MANAGERS, can be added to the SUPERVISOR object. Any object that is security equivalent to an object listed in the MANAGERS property is considered a workgroup manager. The MANAGERS property is "read supervisor" and "write supervisor." If the MANAGERS property does not exist, no objects are considered workgroup managers.

Workgroup managers, like the network supervisor, are allowed to create objects in the bindery. All newly-created objects have a set property called OBJ\_SUPERVISORS. This property is "read object" and "write supervisor" and is initialized to contain a single element: the Object ID of the workgroup manager who created the object. If an object does not have the OBJ\_SUPERVISORS property, the object is assumed to have been created by the network supervisor.

The meaning of supervisor access changes in relation to bindery object access. You are now considered the supervisor of a target object if you are security equivalent to an object that appears in the target object's OBJ\_SUPERVISORS property. An object's supervisor can do anything to the object that the system supervisor can do, including reading and modifying supervisor-level properties, deleting the object, etc.

Workgroup managers have certain rights and abilities beyond those of normal users. They can create new objects in the bindery. They can also control the login restriction properties (such as login times, password length restrictions, and account balances) of the users they create. They cannot, however, control these parameters on their own account (unless they are included in the OBJ\_SUPERVISORS property of their own user object).

Workgroup managers do not have extra rights in the directory tree. They must be granted rights in the standard way by an entity (usually the supervisor) who has sufficient rights to grant access rights to appropriate portions of the directory tree. Like other users, workgroup managers can grant rights only to other users (including users they have created) in directories where they have parental rights.

A user is considered a workgroup manager if the user is security equivalent to any object listed in the MANAGERS property of the supervisor. A user is considered to have supervisor rights over a specific object if the user is security equivalent to anything listed in the object's OBJ\_SUPERVISORS property. Therefore, a user could become a supervisor of a specific object without being a workgroup manager.

The workgroup manager concept is two-tiered: the network supervisor creates workgroup managers, and workgroup managers create and manage users. Workgroup managers cannot create other workgroup managers; the relationship is not hierarchical. The bindery treats workgroup managers as peers. More complex relationships can be created by controlling the places in the

directory tree where workgroup managers are given rights, which effectively creates hierarchical relationships in the data they control.

Workgroup managers supplement, but do not replace, the network supervisor. The network supervisor maintains absolute control over the network. The network supervisor and all objects that are security equivalent to the supervisor can still call the same functions they could before NetWare implemented workgroup managers.

A workgroup manager cannot add an object to a group that is "write supervisor" unless the manager is the supervisor of both the group and the object. This restriction ensures that a workgroup manager cannot create a new user and then make the user security equivalent to the system supervisor. Workgroup managers can do the following:

- ♦ Clear another station if the caller is an object supervisor of the object logged in to the target station.
- ♦ Create new queues and destroy queues they have created.
- ♦ Add users (objects) to a queue only if they are object supervisors of both the queue and the target user.
- ♦ Request certain statistical information (such as the number of directories owned, the number of files owned, and the number of disk blocks used) about any supervised object.
- ♦ Change the supervisor-controlled field in files owned by supervised objects. These fields include the file attributes flags, the extended attributes (including read audit and write audit bits), the 64-byte extended directory information area, and the file owner (if the new target owner is also supervised by the caller).
- ♦ Change the owner of a directory if they supervise both the old directory owner and the proposed new directory owner.



# 8

## NCPs

This section describes each of the Bindery NCPs, their Request and Reply formats, and Return Values.

# Add Bindery Object To Set 0x2222 23 65

Allows a client to add a member (object) to a group property of some object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Decimal	Hex	Description
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (65)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	OldPasswordLen	byte
14 + ObjectNameLen	OldPassword	byte[OldPasswordLen]
14+ObjectNameLen + OldPasswordLen	NewPasswordLen	byte
15 + ObjectNameLen + OldPasswordLen	NewPassword	byte[NewPasswordLen]

## Return Values

Deminal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
197	0xC5	Login Lockout
215	0xD7	Duplicate Password
232	0xE8	Write To Group
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
248	0xF8	No Property Write



Deminal	Hex	Description
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

**NetWare 2.1 and above.** The file server requires the PASSWORD property to be "file server read, file server write" (0x44). Therefore, the file server must create the property. If this request is made by a caller that is the supervisor of the object, **Change Bindery Object Password** directs the file server to create the PASSWORD property. When the PASSWORD property is initially created, the caller can pass a NULL string to *OldPassword*. Objects of any type (such as print servers, job servers, etc.) can have a PASSWORD property.

**NetWare 2.0a and below.** The PASSWORD property is "supervisor read, supervisor write" (0x33). However, clients can still call **Change Bindery Object Password**, even if they are not logged in as the object whose PASSWORD property is being changed because the client would need to know the *ObjectName* and the object's current password ( *OldPassword*) to log in as that object. **Change Bindery Object Password** will fail if the object does not already have a PASSWORD property. The supervisor creates the PASSWORD property at the time the object (user) is created.

Any client can call **Change Bindery Object Password**.

$SubFuncStructLen = 6 + ObjectNameLen + OldPasswordLen + NewPasswordLen$

## See Also

**Change User Password (old) 0x2222 23 01 (page 124), Verify Bindery Object Password 0x2222 23 63 (page 168)**

# Change Bindery Object Password 0x2222 23 64

Allows clients (users) to change an object's (usually their own object's) password.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (64)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PropertyNameLen	byte
14 + ObjectNameLen	PropertyName	byte[PropertyNameLen]
14 + ObjectNameLen + PropertyNameLen	MemberType	word (Hi-Lo)
16 + ObjectNameLen + PropertyNameLen	MemberNameLen	byte
17 + ObjectNameLen + PropertyNameLen	MemberName	byte[MemberNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
232	0xE8	Write To Group
233	0xE9	Member Exists
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard

Decimal	Hex	Description
248	0xF8	No Property Write
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType*, *ObjectName*, and *PropertyName* cannot contain wildcard characters. The name and type of the member object to be added cannot contain wildcard characters either.

*ObjectName*, *PropertyName*, and *MemberName* must exist in the bindery before calling **Add Bindery Object To Set**.

The property must be of type set. If the set property has "write supervisor" security, you must be an object supervisor of both the object and the new member of the set.

If the client has write privileges to the specified property, that client can call **Add Bindery Object To Set**.

$SubFuncStructLen = 8 + ObjectNameLen + PropertyNameLen + MemberNameLen$

## See Also

**Delete Bindery Object From Set 0x2222 23 66 (page 133), Is Bindery Object In Set 0x2222 23 67 (page 148), Scan Bindery Object 0x2222 23 55 (page 160)**

# Change Bindery Object Security 0x2222 23 56

Allows an object supervisor to change the security level mask of an object in the bindery.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (56)	byte
10	NewSecurity	byte
11	ObjectType	word (Hi-Lo)
13	ObjectNameLen	byte
14	ObjectName	byte[ObjectNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
245	0xF5	No Object Create
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

Security levels above level 3 (supervisor) cannot be changed by calling **Change Bindery Object Security**.

Only object supervisors can call **Change Bindery Object Security**.

## See Also

[Get Bindery Access Level 0x2222 23 70 \(page 137\)](#), [Get Bindery Object Access Level 0x2222 23 72 \(page 139\)](#)

# Change Property Security 0x2222 23 59

Allows a client to change the security level mask on a property.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (59)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	NewPropertySecurity	byte
14 + ObjectNameLen	PropertyNameLen	byte
15 + ObjectNameLen	PropertyName	byte[PropertyNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
242	0xF2	No Object Read
246	0xF6	No Property Delete
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1), and *ObjectName* and *PropertyName* cannot contain wildcard characters.

All clients who have property creation privileges for the specified object and property read and write privileges for the specified property can call **Change Property Security**.

A client cannot change a property's security to any level above the security level the client has for that property.

$SubFuncStrucLen = 6 + ObjectNameLen + PropertyNameLen$

## See Also

**Get Bindery Access Level 0x2222 23 70 (page 137), Get Bindery Object Access Level 0x2222 23 72 (page 139)**

# Change User Password (old) 0x2222 23 01

Changes a user's password but does not change the password of any other object type.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (1)	byte
10	UserNameLen	byte
11	UserName	byte[UserNameLen]
11 + UserNameLen	OldPropertyLen	byte
12 + UserNameLen	OldPassword	byte[OldPasswordLen]
12 + UserNameLen + OldPasswordLen	NewPasswordLen	byte
13 + UserNameLen + OldPasswordLen	NewPassword	byte[NewPasswordLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
214	0xD6	Unencrypted Passwords Not Allowed
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
245	0xF5	No Object Create
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure



## Remarks

**Change User Password** (old) is a request from the earliest version of NetWare. It works the same as **Change Bindery Object Password** (0x2222 23 64), except that it assumes an object of type USER.

*SubFuncStrucLen = 4 + UserNameLen + OldPasswordLen + NewPasswordLen*

## See Also

**Change Bindery Object Password 0x2222 23 64 (page 118)**

# Close Bindery 0x2222 23 68

Closes the bindery.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (68)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Hard Failure

## Remarks

When **Close Bindery** completes, the file server keeps the bindery files open but locked to prevent clients from accessing the bindery directly.

Occasionally, bindery files must be closed so that network functions can take place. When the network is backed up, for example, it is important to back up current bindery files as well. Because files must be closed before they can be backed up, **Close Bindery** allows archive backup programs to temporarily take control of the bindery so that bindery files can be closed, then copied or restored.

When the bindery is closed, only the network supervisor or an object that is security equivalent to the supervisor can read from or write to the bindery.

Closing the bindery disables most network functions. Therefore, the time that the bindery is closed should be kept to a minimum.

The bindery will automatically be reopened if the station that closed it disconnects or if the station calls **End Of Job** 0x2222 24 for task zero.

**Close Bindery** will fail if the client is not logged in as the network supervisor or if the bindery is not open.

## See Also

**Open Bindery 0x2222 23 69 (page 155)**

# Create Bindery Object 0x2222 23 50

Creates a bindery object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (6 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (50)	byte
10	StatusFlags	byte
11	SecurityLevel	byte
12	ObjectType	word (Hi-Lo)
14	ObjectNameLen	byte
15	ObjectName	byte[ObjectNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
231	0xE7	No Disk Track
238	0xEE	Object Exists
239	0xEF	Illegal Name
241	0xF1	Bindery Security
245	0xF5	No Object Create
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

Bit 0 of *StatusFlags* should be set if the object being created is dynamic; bit 0 should be cleared if the object being created is static.

**SecurityLevel** should have the low (search) and high (property create) bits appropriately set. A client cannot set security levels to "file server" access.

*ObjectName* can be up to 47 characters long.

Only supervisors of the specified object can call **Create Bindery Object**. Clients in the WORKGROUP\_MANAGERS property of an object are also considered supervisors of the object.

## See Also

**Delete Bindery Object 0x2222 23 51 (page 131)**

# Create Property 0x2222 23 57

Creates a bindery object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (57)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PropertyFlags	byte
14 + ObjectNameLen	PropertySecurity	byte
15 + ObjectNameLen	PropertyNameLen	byte
16 + ObjectNameLen	PropertyName	byte[PropertyName]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
237	0xED	Property Exists
239	0xEF	Illegal Name
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
242	0xF2	No Object Read
246	0xF6	No Property Delete
247	0xF7	No Property Create
251	0xFB	No Such Property

Decimal	Hex	Description
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1), and *ObjectName* and *PropertyName* cannot contain wildcard characters.

Bit 0 of *PropertyFlags* should be set if the property is dynamic; bit 1 of *PropertyFlags* should be set if the property is a set property.

The *PropertySecurity* mask attached to the target object regulates which clients can call **Create Property**.

$SubFuncStructLen = 7 + ObjectNameLen + PropertyNameLen$

## See Also

**Change Property Security 0x2222 23 59 (page 122), Delete Property 0x2222 23 58 (page 135), Scan Property 0x2222 23 60 (page 166)**

# Delete Bindery Object 0x2222 23 51

Removes one object and all its associated properties from the bindery.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (51)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
242	0xF2	No Object Read
244	0xF4	No Object Delete
246	0xF6	No Property Delete
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The object must be fully specified by both its type and name. *ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

Only a client that is an object supervisor of the specified object can call **Delete Bindery Object**. Clients are not allowed to remove objects that have "file server" security levels.

## See Also

[Create Bindery Object 0x2222 23 50 \(page 127\)](#)



# Delete Bindery Object From Set 0x2222 23 66

Removes a member from a set property of the specified object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (66)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PropertyNameLen	byte
14 + ObjectNameLen	PropertyName	byte[PropertyNameLen]
14 + ObjectNameLen + PropertyNameLen	MemberType	word (Hi-Lo)
16 + ObjectNameLen + PropertyNameLen	MemberNameLen	byte
17 + ObjectNameLen + PropertyNameLen	MemberName	byte[MemberNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
235	0xEB	Property Not Set Property
240	0xF0	Illegal Wildcard
248	0xF8	No Property Write
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The object, property, and member are specified by name. *ObjectType* cannot be WILD (-1), and *ObjectName* and *PropertyName* cannot contain wildcard characters.

Clients must have write privileges to the specified property to call **Delete Bindery Object From Set**. If the property security is "write supervisor," the client must supervise both the object being deleted and the object containing the affected set property.

$SubFuncStructLen = 8 + ObjectNameLen + PropertyNameLen + MemberNameLen$

## See Also

**Add Bindery Object To Set 0x2222 23 65 (page 116)**

# Delete Property 0x2222 23 58

Removes one or more properties from the specified object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (58)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PropertyNameLen	byte
14 + ObjectNameLen	PropertyName	byte[PropertyNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
246	0xF6	No Property Delete
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

*PropertyName* can contain wildcard characters. If *PropertyName* contains wildcard characters, all properties that match the name string will be removed from the object.

The object's and property's security level masks determine whether a property can be removed. Properties with security levels above that of the requesting client will not be visible to the client and, therefore, cannot be read, written to, or deleted by the client.

$SubFuncStrucLen = 5 + ObjectNameLen + PropertyNameLen$

## See Also

**Create Property 0x2222 23 57 (page 129)**

# Get Bindery Access Level 0x2222 23 70

Returns the client's current security level mask in the bindery.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (70)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SecurityAccessLevel	byte
9	LoggedObjectID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory

## Remarks

To determine your general bindery rights, look at the two bits of the returned *SecurityAccessLevel* mask.

You also have access privileges (read and write) to the object when logging in. These privileges will not be reflected in the general *SecurityAccessLevel* mask, so the trustee ID number of the object that is used to log in is also returned as the *LoggedClientID*. The client has supervisor access privileges (0x44) to any object that contains the *LoggedClientID* in its OPERATORS property.

Any client can call **Get Bindery Access Level**.

## See Also

[Get Bindery Object Access Level 0x2222 23 72 \(page 139\)](#)

# Get Bindery Object Access Level 0x2222 23 72

Returns the client's bindery access level to the specified object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (72)	byte
10	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectAccessLevel	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory

## Remarks

Get Bindery Object Access Level becomes valid with the introduction of workgroup members. **“Workgroup Managers” on page 112** are given certain privileges to bindery objects by the network supervisor or by other objects that have supervisor privileges to bindery objects.

## See Also

**Get Bindery Access Level 0x2222 23 70 (page 137)**

# Get Bindery Object ID 0x2222 23 53

Allows a client to map an object name to its corresponding unique object ID number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (53)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectID	long (Hi-Lo)
12	ObjectType	word (Hi-Lo)
14	ObjectName	byte[48]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure



## Remarks

*ObjectID* numbers are valid only on the server from which they are extracted since each server maintains its own bindery.

*ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

If the specified object is in the bindery and if the client has read security privileges to the object, that client can call **Get Bindery Object ID**.

## See Also

**Get Bindery Object Name 0x2222 23 54 (page 142)**

# Get Bindery Object Name 0x2222 23 54

Allows a client to map an object ID number to an object name and object type.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (54)	byte
10	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectID	long (Hi-Lo)
12	ObjectType	word (Hi-Lo)
14	ObjectName	byte[48]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
241	0xF1	Bindery Security
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

If an object in the bindery corresponds to the specified *ObjectID* number and if the client has read privileges to that object, that client can call **Get Bindery Object Name**.

## See Also

[Get Bindery Object ID 0x2222 23 53 \(page 140\)](#)

# Get Group Number (old) 0x2222 23 07

Allows a client to map a object name of type GROUP to its corresponding unique object ID number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (54)	byte
10	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectID	long (Hi-Lo)
12	ObjectType	word (Hi-Lo)
14	ObjectName	byte[48]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
241	0xF1	Bindery Security
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

If an object in the bindery corresponds to the specified *ObjectID* number and if the client has read privileges to that object, that client can call **Get Bindery Object Name**.

## See Also

**[Get Bindery Object ID 0x2222 23 53 \(page 140\)](#)**

# Get User Number (old) 0x2222 23 03

Returns the user object ID for the specified user.

**NetWare Server:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2 + UserNameLen)	word (Hi-Lo)
9	SubFunctionCode (3)	byte
10	UserNameLen	byte
11	UserName	byte[UserNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	UserObjectID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*UserName* must be an object of type USER. The *UserObjectID* that is returned will be stamped on any directories and file that the object (user) creates. You can use the *UserObjectID* from **Add Trustee To Directory** (0x2222 22 13) to give the object access rights to directories.

**Get User Number (old)** is obsolete and maintained for compatibility only with the earliest version of NetWare. Call **Get Bindery Object Number** (0x2222 23 53) instead.

# Is Bindery Object In Set 0x2222 23 67

Allows a client to check whether an object appears as a member in a set property.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (67)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PropertyNameLen	byte
14 + ObjectNameLen	PropertyName	byte[PropertyNameLen]
14 + ObjectNameLen + PropertyNameLen	MemberType	word (Hi-Lo)
16 + ObjectNameLen + PropertyNameLen	MemberNameLen	byte
17 + ObjectNameLen + PropertyNameLen	MemberName	byte[MemberNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked



Decimal	Hex	Description
255	0xFF	Hard Failure

## Remarks

The member being examined, the property, and the object that owns the property must all be specified by name and type. *ObjectType* cannot be WILD (-1), and *ObjectName* and *PropertyName* cannot contain wildcard characters.

If *ObjectNameLen* is greater than 47 or if the *PropertyNameLen* is greater than 15, **Is Bindery Object In Set** will fail.

Any client with read security privileges for the set property can call **Is Bindery Object In Set**.

$SubFuncStrucLen = 8 + ObjectNameLen + PropertyNameLen + MemberNameLen$

## See Also

**Add Bindery Object To Set 0x2222 23 65 (page 116)**

# Is Calling Station a Manager 0x2222 23 73

Checks to see if the calling station has manager privileges in the bindery.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (73)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Calling Station is a manager
255	0xFF	Calling Station is not a manager

## Remarks

A station is a manager if it is a supervisor or if it appears in the MANAGERS property of the supervisor object.

## See Also

**Scan Property 0x2222 23 60 (page 166), Read Property Value 0x2222 23 61 (page 156)**

# Keyed Change Password 0x2222 23 75

Changes the password for a bindery object that requires keyed login.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (75)	byte
10	Key	byte[8]
18	ObjectType	word (Hi-Lo)
20	ObjectNameLen	byte
21	ObjectName	byte[ObjectNameLen]
22 + ObjectNameLen	NewPasswordLen	byte
23 + ObjectNameLen	NewPassword	byte[NewPasswordLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
197	0xC5	Account Locked
254	0xFE	Bindery Locked
255	0xFF	Password Verification Failed

## See Also

**Keyed Verify Password 0x2222 23 74 (page 152)**

# Keyed Verify Password 0x2222 23 74

Verifies a password for a bindery object that requires keyed login.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (12 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (74)	byte
10	Key	byte[8]
18	ObjectType	word (Hi-Lo)
20	ObjectNameLen	byte
21	ObjectName	byte[ObjectNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
197	0xC5	Account Locked
254	0xFE	Bindery Locked
255	0xFF	Password Verification Failed

## See Also

**Keyed Change Password 0x2222 23 75 (page 151)**

# List Relations Of an Object 0x2222 23 76

Returns a list of up to 32 bindery object IDs (relations) that are in a set property of the specified bindery object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (76)	byte
10	LastSeen	long (Hi-Lo)
14	ObjectType	word (Hi-Lo)
16	NameLen	byte
17	ObjectName	byte[NameLen]
17 + NameLen	PropertyLen	byte
18 + NameLen	PropertyName	byte[PropertyLen]

## Reply Format

Offset	Content	Type
Reply header		
8	Count	word (Hi-Lo)
10	Relations	long[Count] (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
240	0xF0	Illegal Wildcard
242	0xF2	No Object Read Rights
254	0xFE	Directory Locked

Decimal	Hex	Description
255	0xFF	Hard Failure

## Remarks

*LastSeen* should be initialized to -1. If the returned *Count* value is 32, a subsequent request should be made to determine if there are more relations. On subsequent requests, *LastSeen* should be set to the last bindery ID in the *Relations* array.

$SubFuncStrucLen = 9 + NameLen + PropertyLen$

# Open Bindery 0x2222 23 69

Reopens the bindery files that have been closed with **Close Bindery** (0x2222 23 68).

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (69)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

Occasionally, bindery files must be closed so that network functions can take place. When the network is backed up, for example, it is important to back up current bindery files as well. Because files must be closed before they can be backed up, **Close Bindery** allows archive backup programs to temporarily take control of the bindery so that bindery files can be closed, then copied or restored.

Once the backup is complete, the bindery can be reopened by calling **Open Bindery**.

When the bindery is closed, only the network supervisor or an object that is security equivalent to the supervisor can read from or write to the bindery.

The bindery will automatically be reopened if the station that closed it disconnects or if the station calls **End Of Job** 0x2222 24 for task zero.

If the client is not logged in as the network supervisor or if the bindery is already open, **Open Bindery** will fail.

## See Also

**Close Bindery 0x2222 23 68 (page 126)**

# Read Property Value 0x2222 23 61

Allows a client to retrieve the value associated with the specified property.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (61)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	SegmentNumber	byte
14 + ObjectNameLen	PropertyNameLen	byte
15 + ObjectNameLen	PropertyName	byte[PropertyNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	PropertyValue	byte[128]
136	MoreFlag	byte
137	PropertyFlags	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
150	0x96	Server Out of Memory



Decimal	Hex	Description
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
249	0xF9	No Property Read
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

Property values are stored in a single 128-byte segment. *SegmentNumber* must be set to 1 to read the first segment of a value; it must be incremented by one for each subsequent call. **Read Property Value** returns the 128-byte segment corresponding to *SegmentNumber*.

If the requested *SegmentNumber* is not the last segment in the value, *MoreFlag* is 0xFF. If *SegmentNumber* is the last segment, *MoreFlag* is zero.

*PropertyFlags* contains the property's status flag. This byte can be tested to determine whether the property is a set property or an item property and whether the property is static or dynamic.

*ObjectType* cannot be WILD (-1), and *ObjectName* and *PropertyName* cannot contain wildcard characters.

Any client with read privileges to the specified property can call **Read Property Value**.

$SubFuncStructLen = 6 + ObjectNameLen + PropertyNameLen$

## See Also

**Is Calling Station a Manager 0x2222 23 73 (page 150), Scan Property 0x2222 23 60 (page 166)**

# Rename Object 0x2222 23 52

Renames a bindery object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (52)	byte
10	ObjectType	word (Hi-Lo)
12	OldObjectNameLen	byte
13	OldObjectName	byte[OldObjectNameLen]
13 + OldObjectNameLen	NewObjNameLen	byte
14 + OldObjectNameLen	NewObjectName	byte[NewObjNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
238	0xEE	Object Exists
240	0xF0	Illegal Wildcard
243	0xF3	No Object Rename
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

Wildcard characters are not allowed in either the old or new name specifications.

*ObjectType* cannot be WILD (-1) and must agree with the object type recorded in the bindery.

Only an object supervisor can call **Rename Object**.

$$SubFuncStrucLen = 5 + OldObjectNameLen + NewObjNameLen$$

# Scan Bindery Object 0x2222 23 55

Allows a client to scan the server's bindery to determine what objects exist.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (8 + SearchNameLen)	word (Hi-Lo)
9	SubFunctionCode (55)	byte
10	LastObjectSeen	long (Hi-Lo)
14	SearchObjectType	word (Hi-Lo)
16	SearchNameLen	byte
17	SearchObjectName	byte[SearchNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectID	long (Hi-Lo)
12	ObjectType	word (Hi-Lo)
14	ObjectName	byte[48]
62	ObjectFlags	byte
63	ObjectSecurity	byte
64	ObjectHasProperties	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
239	0xEF	Illegal Name

Decimal	Hex	Description
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*SearchObjectType* can be WILD (-1) or it can be set to find only a specific type of object.

*ObjectSearchName* can contain wildcard characters.

*LastObjectSeen* should initially be set to -1L (long -1). If you want to iteratively search through the server's bindery, set *LastObjectSeen* to the *ObjectID* returned by the server with the previous request.

*ObjectHasProperties* will always be set because all objects have a bindery level property that is used internally.

If the object exists in the bindery and if the client has search privileges to the object, any client can call **Scan Bindery Object**.

$SubFuncStrucLen = 6 + ObjectNameLen + PropertyNameLen$

## See Also

**Is Bindery Object In Set 0x2222 23 67 (page 148), Delete Bindery Object From Set 0x2222 23 66 (page 133), Add Bindery Object To Set 0x2222 23 65 (page 116), Scan Bindery Object Trustee Paths 0x2222 23 71 (page 164), Create Bindery Object 0x2222 23 50 (page 127), Delete Bindery Object 0x2222 23 51 (page 131)**

# Scan Bindery Object (List) 0x2222 23 32

Allows a client to scan the server's bindery to determine what objects exist.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (14 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (32)	byte
10	NextObjectID	long (Hi-Lo)
14	ObjectType	long (Hi-Lo)
18	InfoFlags	long (Lo-Hi)
22	ObjectNameLen	byte
23 + ObjectNameLen	ObjectName	byte[ObjectNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectInfoRtnCount	long (Lo-Hi)
12	NextObjectID	long (Hi-Lo)
16+	ObjectInfoStruc (page 174)	Structure

## Parameters

### *InfoFlags*

(Request) Specifies which object information is to be returned:

0x10000000 RtnObjectSecurity

0x20000000 RtnObjectFlags

0x40000000 RtnObjectType

0x80000000 RtnObjectName

### *ObjectNameLen*

(Request) Specifies the length of *ObjectName*.

*ObjectName*

(Request) Specifies the name of the object to search for (can contain wildcard characters).

*ObjectInfoRtnCount*

(Reply) Specifies the number of *ObjectInfoStrucs* that are returned.

*NextObjectID*

(Reply) Specifies the next ID value to request if more objects exist than could be returned in the reply buffer. If *NextObjectID* equals -1, there are no more objects available.

*ObjectInfoStruc*

(Reply) Points to the *ObjectInfoStruct*, which contains the object information.

*ObjectInfoRtnCount*

(Reply) Specifies the number of returned structures.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
239	0xEF	Illegal Name
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

**Scan Bindery Object (List)** returns a list of all objects that match the search criteria.

Set *NextObjectID* initially to -1L (long -1). For subsequent requests, set *NextObjectID* to the *NextObjectID* value returned by the server with the previous request to iteratively search through the server's bindery.

*ObjectType* can be WILD (-1) or can be set to find only a specific type of object.

If the object exists in the bindery and if the client has search privileges to the object, that client can call **Scan Bindery Object (List)**.

## See Also

**Is Bindery Object In Set 0x2222 23 67 (page 148), Delete Bindery Object From Set 0x2222 23 66 (page 133), Add Bindery Object To Set 0x2222 23 65 (page 116), Scan Bindery Object Trustee Paths 0x2222 23 71 (page 164), Create Bindery Object 0x2222 23 50 (page 127), Delete Bindery Object 0x2222 23 51 (page 131), Scan Bindery Object 0x2222 23 55 (page 160)**

# Scan Bindery Object Trustee Paths 0x2222 23 71

Returns a path on the specified volume where the given object ID is a trustee.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (8)	word (Hi-Lo)
9	SubFunctionCode (71)	byte
10	VolumeNumber	byte
11	LastSequenceNumber	word (Hi-Lo)
13	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	NextSequenceNumber	word (Hi-Lo)
10	ObjectID	long (Hi-Lo)
14	TrusteeAccessMask	byte
15	DirectoryPathLen	byte
16	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
147	0x93	No Read Privileges
150	0x96	Server Out of Memory
161	0xA1	Directory I/O Error
240	0xF0	Illegal Wildcard



Decimal	Hex	Description
241	0xF1	Bindery Security
242	0xF2	No Object Read
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The path that is returned is also known as the trustee path. **Scan Bindery Object Trustee Paths** can be called iteratively to determine all the paths on a specified volume where *ObjectID* appears in a directory's trustee list.

You should set *VolumeNumber* to the desired volume number. To map a volume name to its mount number, or to determine which volumes are mounted on a server, call **Get Volume Number** (0x2222 22 5) and **Get Volume Name** (0x2222 22 6).

Set *SequenceNumber* initially to -1L (long -1). For subsequent requests, *SequenceNumber* should be set to the value returned in *NextSequenceNumber* of the previous request.

*ObjectID* must contain the number of the object (user) whose trustee directory paths you are requesting.

The file server returns *DirectoryPath* in the following format:

```
volume:directory/subdirectory/subdirectory/. . .
```

Directory paths are not returned in any special order. The server will return a *DirectoryPathLen* of zero if all directory paths where the object is listed as a trustee have been returned.

A client who is supervisor (or equivalent) can call **Scan Bindery Object Trustee Paths** to investigate the trustee directory rights of any object in the bindery.

Any client can call **Scan Bindery Object Trustee Paths** to determine its own trustee paths, or the trustee paths of any object (up to 32) to which the client is security equivalent.

## See Also

**Scan Bindery Object 0x2222 23 55 (page 160)**

# Scan Property 0x2222 23 60

Allows a client to scan the properties that are attached to a bindery object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (60)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	LastInstance	long (Hi-Lo)
17 + ObjectNameLen	PropertyNameLen	byte
18 + ObjectNameLen	SearchPropertyName	byte[PropertyNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	PropertyName	byte[16]
24	PropertyFlags	byte
25	PropertySecurity	byte
26	SearchInstance	long (Hi-Lo)
30	ValueAvailable	byte
31	MoreProperties	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
150	0x96	Server Out of Memory
240	0xF0	Illegal Wildcard
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*ObjectType* cannot be WILD (-1); *ObjectName* and *PropertyName* cannot contain wildcard characters.

Set *LastInstance* initially to -1L (long -1). For subsequent requests to scan an object's properties, set *LastInstance* to the *SearchInstance* returned by the previous request.

If the property does not have an associated value, *ValueAvailable* is zero. If the property has a value, *ValueAvailable* is -1 (0xFF).

If the property is the last property associated with the object, *MoreProperties* is zero. If there are other properties associated with the object, *MoreProperties* is -1 (0xFF).

A client who has read privileges for both the target object (*ObjectName*) and the target property (*SearchPropertyName*) can call **Scan Property**.

$SubFuncStrucLen = 9 + ObjectNameLen + PropertyNameLen$

## See Also

**Create Property 0x2222 23 57 (page 129), Is Calling Station a Manager 0x2222 23 73 (page 150), Read Property Value 0x2222 23 61 (page 156)**

# Verify Bindery Object Password 0x2222 23 63

Allows other servers or interested clients to verify the identity of an object (user) or to check the validity of a password.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (63)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	PasswordLen	byte
14 + ObjectNameLen	Password	byte[PasswordLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
197	0xC5	Login Lockout
232	0xE8	Write To Group
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

Passwords are defined as strings of up to 127 bytes. The password terminates at the first NULL byte that is encountered in the password string.

*ObjectType* cannot be WILD (-1), and *ObjectName* cannot contain wildcard characters.

If an object has no password property, the server does not allow clients to identify themselves as that object in a Login Object request. If an object has an attached PASSWORD property but the PASSWORD property is NULL, the server considers the PASSWORD as a match for a NULL string and allows clients to log in using that *ObjectName*.

Any client can call **Verify Bindery Object Password**.

$SubFuncStrucLen = 5 + ObjectNameLen + PasswordLen$

## See Also

**Change Bindery Object Password 0x2222 23 64 (page 118)**

# Write Property Value 0x2222 23 62

Allows a client to write a value to an item property.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (62)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]
13 + ObjectNameLen	SegmentNumber	byte
14 + ObjectNameLen	MoreFlag	byte
15 + ObjectNameLen	PropertyNameLen	byte
16 + ObjectNameLen	PropertyName	byte[PropertyNameLen]
16 + ObjectNameLen + PropertyNameLen	PropertyValue	byte[128]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out of Memory
232	0xE8	Write To Group
236	0xEC	No Such Set
240	0xF0	Illegal Wildcard
248	0xF8	No Property Write
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked

Decimal	Hex	Description
255	0xFF	Hard Failure

## Remarks

Values of set properties are manipulated by calling **Add Bindery Object To Set** (0x2222 2 65) and **Delete Bindery Object From Set** (0x2222 23 66).

Property values must be within a single 128-byte segment.

**Write Property Value** writes out the 128-byte value segment to the specified *SegmentNumber*. The first segment of a value is segment 1. Before segment *n* can be written, segments 1 to *n*-1 must be written. Once a property value has been established, segments of it can be read or written in random order.

The *More* flag informs the bindery whether the written segment is the last segment of the value. If *More* is zero and the current value has segments beyond the segment being written, the bindery will truncate *PropertyValue* and discard the extra segments.

Any client that has write privileges to the target property (*PropertyName*) can call **Write Property Value**.

$SubFuncStrucLen = 135 + ObjectNameLen + PropertyNameLen$

## See Also

**Read Property Value 0x2222 23 61 (page 156)**





# 9

## Structures

This section describes the Bindery structures and their fields.

# ObjectInfoStruc

Returns the specified object information.

## Syntax

```
struct ObjectInfoStruc
{
    LONG    ObjectIDInfo;
    LONG    ObjectType;
    BYTE    ObjectSecurity;
    BYTE    ObjectName[];
};
```

## Fields

ObjectIDInfo

(Hi-Lo) Is always returned, regardless of the value passed to *InfoFlags*.

ObjectType

(Hi-Lo) Is returned if *InfoFlags* specifies this information.

ObjectSecurity

Is returned if *InfoFlags* specifies this information.

ObjectName

(ASCIIZ) Is returned if *InfoFlags* specifies this information.

# V

## Connection

Connection NCPs allow a client and a server to create, maintain, and destroy a service connection. Some Connection NCPs allow a client and server to interact reliably when using an unreliable (connectionless) datagram service to communicate. Connection maintenance is necessary in an unreliable messaging environment. When reliable message-switching network interfaces are used, most of connection maintenance is unnecessary.

To access Connection NCPs, use the following:

- ♦ [Chapter 10, “Concepts,” on page 177](#)
- ♦ [Chapter 11, “NCPs,” on page 179](#)
- ♦ [Chapter 12, “Structures,” on page 221](#)



# 10 Concepts

This section explains ideas that are common to Connection NCPs.

## Unreliable Messaging

In an unreliable message interface environment, the file server checks its current list of service connections when it receives a **Create Service Connection** request. If the server already has a connection with the client issuing the request, that connection is reinitialized to a startup state and reused.

When any other service request arrives, the server first determines whether the request sequence number matches the sequence number of the preceding service response message. (The server always keeps a copy of the service response that it most recently sent to each client it is servicing.) If the sequence numbers do not match, the service request is processed to generate a new service response message, and the old service response is replaced. This newly created response message is then sent to the client.

With an unreliable message interface, the client must retry a given service request until it either receives the corresponding service response or until the file server or the connecting network has failed. The client must retry if a service message is lost enroute from the client to the server, or the service response is lost enroute from the server to the client.

## Timeout Period

The client must determine what a "reasonable" amount of time is for waiting between retries; this timeout period must allow time for a round-trip message exchange and request processing. One way to determine the timeout period is for the client to maintain a weighted average of the times required by previous requests.

Because the time required to service a message may vary, depending on the request type and the server's workload, it is possible for a server to receive a duplicate of the request it is currently servicing. This event signals to the server that the client's timeout has expired.

In such cases, the server might reply to the client with a special Request Being Processed message that informs the client that its request has been received by the server and is being acted upon. When a client receives the Request Being Processed message, it must reset its timeout and retry counters and resume waiting for its actual service response message.

If the service response does not arrive before the client's timeout expires, the client must repeat the service request to safeguard against the possibility of a lost server response. If the server is still servicing the request, it will respond with another Request Being Processed message. In this manner, the server can cause a client to wait indefinitely until the client's request has been processed, and the client receives continuing assurances that the communications link is functioning and the server is alive.



# 11

## NCPs

This section describes each of the Connection NCPs, their Request and Reply formats, and Return Values.

# Change Connection State 0x2222 23 29

NetWare Server: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunction (29)	byte
10	RequestCode	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
122	0x7A	Connection Already Temporary
123	0x7B	Connection Already Logged In
124	0x7C	Connection Not Authenticated
224	0xE0	No Login Connections Available
251	0xFB	Unknown Request
253	0xFD	Bad Station Number

## Remarks

*RequestCode* can have the following values:

- 0 Change logged in to temporary authenticated
- 1 Change temporary authenticated to logged in
- Other Invalid (ERR\_UNKNOWN\_REQUEST)



# Clear Connection Number 0x2222 23 254

Logs out the specified station.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunction (254)	byte
10	ConnectionNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

**Clear Connection Number** is similar to **Clear Connection Number (old)** (0x2222 23 210), but the new request allows station numbers greater than 256.

Any resources of file that the station is using when it is logged out are released and the station's service connection is broken when **Clear Connection Number** completes. The station must reestablish a service connection and log in before it can resume work of the file server.

If the calling station lacks supervisor privileges, No Console Rights (0xC6) is returned and the target station is not affected.

# Create Service Connection 0x1111

Creates a connection between the server and a client.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
0	RequestType (0x1111)	word (Hi-Lo)
2	SequenceNumber (0)	byte
3	ConnectionNumber (0xFF)	byte
4	TaskNumber (ignored)	byte
5	Reserved (ignored)	byte
6	RequestCode (ignored)	byte

## Reply Format

Offset	Content	Type
0	ReplyType (0x3333)	word (Hi-Lo)
2	SequenceNumber (0)	byte
3	ConnectionNumber (NewConnectionNumber)	byte
4	TaskNumber (0)	byte
5	Reserved (0)	byte
6	CompletionCode	byte
7	ConnectionStatus (StatusFlags)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Create Service Connection** will succeed if the server is still logically up and has service connections available.

If the server already has a connection allocated to the specified client, that connection is reinitialized and reused. The client must set the *ConnectionNumber* of all subsequent request messages to the connection number ( *NewConnectionNumber*) returned by the server in the response message. Also, the *SequenceNumber* of the next service request must be set to 1, and the *SequenceNumber* of each succeeding request must be set to 1 plus the *SequenceNumber* of the previous successful request message.

Bits 2 and 4 of *ConnectionStatus* must be checked by the client each time that the client receives a service response. Bit 2 is set if the server has no further connections available. Bit 4 is set if the server is down.

The server can receive a **Create Service Connection** request in three circumstances:

1. A station makes an initial request to create a service connection.

The server does not have a connection allocated to the station. The server services the request by creating a connection and returning a response that contains the new connection number.

2. A station has not yet received a response to the previous request to create a connection.

The server may be in the process of servicing the request, or the server may have already completed servicing the request and returned the response. The station, however, has probably timed out waiting for the response and repeated the request to create a service connection. Since the server knows it has already created the connection, it sends back a response with the same information as the previous response. The server could ignore a duplicate request to create a service connection, but the original response may have been lost, so it is important that a station receive a response to all **Create Service Connection** requests.

3. A station has not destroyed a previously created service connection.

The station that has already created a service connection did not destroy the connection conventionally by calling **Destroy Service Connection**. This situation occurs, for example, when a station is being rebooted or powered off. If a station is rebooted and subsequently sends a **Create Service Connection** request, the server reuses the previous connection number (if the server has not already relinquished the connection).

The server detects whether a station is still using a connection by implementing a watchdog process. The watchdog process is awakened periodically by the server process to check the activity of the current connections. If there are connections that show no activity during the period, the watchdog process attempts to query those stations in a peer-to-peer fashion. With NetWare's watchdog implementation, an IPX packet that requests a simple response is sent to each of the apparent inactive stations. If a station is still using the connection, the station responds appropriately and the connection remains valid. If a negative response or no response is received by the server, the server reinitializes the connection number and makes it available again to any station.

A connection number is simply an index to a table maintained by the server. The table contains, among other items, the unique node address of each station for which a connection is allocated and the socket on which the station is listening for service response packets. Other special-purpose sockets are opened by a station when it creates a service connection. These special-purpose sockets, which include the socket on which a station listens for watchdog packets, are numbered sequentially on the socket dedicated for service response packets. The number of the socket

dedicated to listening for watchdog packets is one plus the number of the socket listening for service responses. The packet structure of the watchdog packet follows:

Offset	Contents	Type
0	Header	IPXPacketHeader
30	PacketSlot	byte
31	SignatureChar	byte

The server sets up the Header so that the packet will be sent to the station in question. *PacketSlot* is set to the connection number, and *SignatureChar* is set to a question mark. If the station that receives the packet must keep the connection contained in *PacketSlot*, the station sets up a similar response packet to return to the server. The station sets *SignatureChar* to Y (for Yes), sets *PacketSlot* to the connection number, and sets up the Header so that the response packet will be sent to the server. When the watchdog process receives this packet, it will not relinquish the corresponding connection.

## See Also

**Destroy Service Connection 0x5555 00 (page 185)**

# Destroy Service Connection 0x5555 00

Causes a server to destroy the specified connection (*ConnectionNumber*) between the server and the client issuing the request.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
0	RequestType (0x5555)	word (Hi-Lo)
2	SequenceNumber (LastSequence + 1)	byte
3	ConnectionNumber (ServiceConnection)	byte
4	TaskNumber (ignored)	byte
5	Reserved (ignored)	byte
6	RequestCode (ignored)	byte

## Reply Format

Offset	Content	Type
0	ReplyType (0x3333)	word (Hi-Lo)
2	SequenceNumber (RequestSequenceNumber)	byte
3	ConnectionNumber (ServiceConnection)	byte
4	TaskNumber (ClientTaskNumber)	byte
5	Reserved (0)	byte
6	CompletionCode (See below)	byte
7	ConnectionStatus (StatusFlags)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

If **Destroy Service Connection** is successful, the connection slot is reinitialized or made available to clients who subsequently request a service connection. If the call fails (because the server has no such connection, or because the specified connection is being used by a client other than the client issuing the request), the server does not reinitialize the connection, but it still sends a service response to the requesting client.

The server is required to send a service response because this service request could be a retransmission of a former request already received and acted upon by the server. If the server's former response was lost in its route to the previous client, the server could have already granted the connection to a new client by the time the old client retries the request.

## See Also

**Create Service Connection 0x1111 (page 182)**

# End Of Job 0x2222 24

Allows a client to inform the server that one of its tasks has finished execution.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (24)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The server will close any active files, unlock any locked byte ranges, and perform the required clean-up for any file server resources that the indicated task has been using. The server does not assume any hierarchical relationships between a client's tasks. If such a relationship exists, the client must ensure that this request is made for each affected task in the appropriate order.

If the task number is set to zero, all the client's tasks have ceased execution and a general "end of job" clean-up should be performed.

# Generate GUIDs 0x2222 23 33

Returns a list of GUIDs from the server.

**NetWare Server:** 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (33)	byte
10	ReturnInfoCount	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ReturnInfoCount	long (Lo-Hi)
12	GUID	byte[ReturnInfoCount][16] (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The GUID is 16 bytes long, and a maximum of 33 can be returned.



# Get Big Packet NCP Max Packet Size 0x2222 97

NetWare Server: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (97)	byte
7	ProposedMaxSize	word (Hi-Lo)
9	SecurityFlag	byte

## Reply Format

Offset	Content	Type
Reply header		
8	AcceptedMaxSize	word (Hi-Lo)
10	EchoSocket	word (Hi-Lo)
12	SecurityFlag	byte

## Parameters

### *SecurityFlag*

(Request) Specifies the appropriate bits:

- 0000001b    ChecksummingRequested
- 00000010b    SignatureRequested
- 0000100b    CompletSignaturesRequested
- 00001000b    EncryptionRequested
- 10000000b    LargeInternetPacketDisable

### *AcceptedMaxSize*

(Reply) Specifies the smallest of the following variables:

- ProposedMaxSize
- Server's global MaxPhysicalPacketSize
- Receiving board's MLIDMaxPacketSize
- Size of two cache buffers

### *EchoSocket*

(Reply) Is used to hold ECHO packets.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
nonzero		NCP 97 not supported

## Remarks

In the reply packet, *SecurityFlag* will be zero if the request packet's *SecurityFlag* bits are consistent with the server's *SignatureLevelRequired* parameter.

If the request packet's *SecurityFlag* bits are not consistent with the server's *SignatureLevelRequired* parameter, the reply packet's *SecurityFlag* bits will be set to indicate which bits the requester should set the next time it calls **Get Big Packet NCP Max Packet Size**.

For example, if the server requires signatures, but the requester issues NCP 97 with *SecurityFlag* set to 00000000b (zero), the reply packet will contain a security flag set to 00000010b (two, which indicates that the server requires signatures). The request should OR the *SecurityFlag* of the reply packet with the *SecurityFlag* in the request packet and make the NCP 97 request again.

**IMPORTANT:** After successfully negotiating the security flags, the requester should store the result of its negotiation in a global variable that indicates whether the NCP session is subject to MD4 signature verification.

The server can be set to one of the following three checksum states:

No Checksums-Checksums are not enabled

Allow Checksums-Client can choose to checksum

Require Checksums-Server will talk only to clients requesting checksums

# Get Connection List From Object 0x2222 23 31

Returns a list of connections using the ObjectID.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (31)	byte
10	ObjectID	long (Hi-Lo)
14	SearchConnNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ConnListLen	word (Lo-Hi)
10	ConnList	long[] (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

[Get Object Connection List 0x2222 23 27 \(page 196\)](#), [Get Object Connection List \(old\) 0x2222 23 21 \(page 197\)](#)

# Get Internet Address 0x2222 23 26

Allows the use of the high connection byte in the Request/Reply packet header.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (26)	byte
10	TargetConnection	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	<b>NetworkAddressStruct</b>	struct
20	ConnectionType	byte

## Parameters

*NetworkAddressStruct*

(Reply) Specifies the network address of the target connection number.

*ConnectionType*

(Reply) Specifies the type of connection the target has:

- 0 Not in use
- 2 NCP
- 11 UDP (for IP)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Get Internet Address (old) 0x2222 23 19

Returns the physical network address of the client workstation when given the workstation's logical connection number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFuncCode (19)	byte
10	TargetConnection	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NetworkAddressStruct	struct

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Get Internet Address (old)** can be called to determine the address of any client that has a connection on the network. Typically, this request is made before setting up a direct network dialogue between two clients.

The format of NetworkAddress is network dependent.

**Get Internet Address (old)** is replaced by **Get Internet Address (0x2222 23 26)**.

## See Also

[Get Object Connection List \(old\) 0x2222 23 21 \(page 197\)](#), [Get Internet Address 0x2222 23 26 \(page 192\)](#)

# Get Login Key 0x2222 23 23

Returns an 8-byte password key for the calling station.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (23)	byte
10	TargetConnection	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Key	byte[8]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	No Allocation Space

## See Also

**Keyed Object Login 0x2222 23 24 (page 207)**

# Get Object Connection List 0x2222 23 27

Allows the use of the high connection byte in the request/reply packet header.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (8 + ObjectNameLen)	word (Hi-Lo)
9	SubFunction (27)	byte
10	SearchConnNumber	long (Lo-Hi)
14	ObjectType	word (Hi-Lo)
16	ObjectNameLen	byte
17	ObjectName	byte[ObjectNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ConnListLen	byte
9	ConnNumberList	long[ConnListLen] (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*SearchConnNumber* should be set to zero initially. Subsequent requests use the highest connection number returned in *ConnNumberList*.

## See Also

**Get Object Connection List (old) 0x2222 23 21 (page 197)**



# Get Object Connection List (old) 0x2222 23 21

Returns a list of the connection numbers on the server for clients logged in with the specified *ObjectName* and *ObjectType*.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + ObjectNameLen)	word (Hi-Lo)
9	SubFunction (21)	byte
10	ObjectType	word (Hi-Lo)
12	ObjectNameLen	byte
13	ObjectName	byte[ObjectNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ListLen	byte
9	ConnectionNumberList	long[ListLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
240	0xF0	Illegal Wildcard
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

If no client is logged in using the specified *ObjectName* and *ObjectType*, the list length returned by the server is set to zero.

**Get Object Connection List (old)** is replaced by **Get Object Connection List** (0x2222 23 27) and replaces **Get User Connection List** (0x2222 23 2).

## See Also

**Get Internet Address (old) 0x2222 23 19 (page 193), Get User Connection List (old) 0x2222 23 02 (page 205), Get Object Connection List 0x2222 23 27 (page 196)**

# Get Station Number 0x2222 19

Returns the calling station's 3-byte station number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (19)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	StationNumber	byte[3]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Get Station's Logged Info (old) 0x2222 23 22 (page 201)**

# Get Station's Logged Info 0x2222 23 28

Allows the use of the high connection byte in the request/reply packet header.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunction (28)	byte
10	TargetConnectionNum	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	UniqueUserID	long (Hi-Lo)
12	UserType	word (Hi-Lo)
14	UserName	byte[48]
62	LoginTime	byte[7]
69	reserved (0)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Get Station's Logged Info (old) 0x2222 23 22

Returns information about a user at the given connection number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (22)	byte
10	TargetConnectionNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	UniqueUserID	long (Hi-Lo)
12	UserType	word (Hi-Lo)
14	UserName	byte[48]
62	LoginTime	byte[7]
69	Reserved	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
252	0xFC	No Such Object
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

*LoginTime* has the following format:

Byte	Value	Description
0	Year	Contains a value from 0-99. Values from 80-99 correspond to the years 1980-1999. Values from 0-79 correspond to the years 2000-2079.
1	Month	Contains a value from 1-12, which corresponds to the months January through December.
2	Day	Contains a value from 1-31, which indicates the current day of the month.
3	Hour	Contains a value from 0-23, which indicates the hour of the day (0 means 12:00 midnight, 23 means 11:00 p.m., etc.).
4	Minute	Contains a value from 0-59, which indicates the current minute in the hour.
5	Second	Contains a value from 0-59, which indicates the current second in the minute.
6	DayOfWeek	Contains a value from 0-6, which corresponds with the weekday (0 for Sunday, 6 for Saturday, etc.).

Any client can call **Get Station's Logged Info (old)** (0x2222 23 22).

**Get Station's Logged Info (old)** (0x2222 23 22) is replaced by **Get Station's Logged Info** (0x2222 23 28).

## See Also

**Get Station's Logged Info 0x2222 23 28 (page 200), Get Object Connection List (old) 0x2222 23 21 (page 197)**

# Get Station's Logged Info (old) 0x2222 23 05

Returns a station's log record from the network to the calling station.

NetWare Server: 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (5)	byte
10	TargetConnectionNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	UserName	byte[16]
24	LoginTime	byte[7]
31	FullName	byte[39]
70	UserID	long
74	SecurityEquivalentList	long[32]
202	Reserved2	byte[64]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
252	0xFC	No Such Object
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The log record includes the name and security equivalence list for the station. The caller must be an object of type USER.

The reply message, not including the reply header, is a fixed length of 258 bytes. *UserName* will contain only the first 15 bytes of the (potential) 47-byte username; all other characters are lost.

*SecurityEquivalentList* contains the ID numbers of up to 32 objects (users and groups) that security equivalent to the caller, while *FullName* and *Reserved2* are set to NULL. *LoginTime* and *UserID* are correct.

**Get Station's Logged Info (old)** (0x2222 23 05) is maintained only for compatibility with the logout process of NetWare 4.61 and below. For NetWare 1.0 through 3.1, **Get Station's Logged Info (old) (0x2222 23 05)** is replaced by **Get Station's Logged Info** (0x2222 23 22). For NetWare 3.11, **Get Station's Logged Info (old)** (0x2222 23 05) is replaced by **Get Station's Logged Info** (0x2222 23 28).

The replacement request should be used to get the names of users logged in to a station, and the bindery requests should be used to get information about a user's groups and security information.

## See Also

**Get Station's Logged Info (old) 0x2222 23 22 (page 201), Get Station's Logged Info 0x2222 23 28 (page 200)**



# Get User Connection List (old) 0x2222 23 02

Returns a list of connection numbers on the server for clients logged in with the specified *UserName* .

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2 + UserNameLen)	word (Hi-Lo)
9	SubFunctionCode (2)	byte
10	UserNameLen	byte
11	UserName	byte[UserNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ListLen	byte
9	ConnectionNumberList	byte[ListLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
240	0xF0	Illegal Wildcard
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

If no client is logged in using the specified UserName, the list length returned by the server is set to zero.

**Get User Connection List (old)** is a request from the earliest version of NetWare. It is the same as **Get Object Connection List** (0x2222 23 21), except that it assumes an object of type USER.

## See Also

**Get Object Connection List (old) 0x2222 23 21 (page 197)**

# Keyed Object Login 0x2222 23 24

Logs in the specified bindery object, using the 8-byte key previously assigned to the object.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (12 + ObjectNameLen)	word (Hi-Lo)
9	SubFunctionCode (24)	byte
10	Key	byte[8]
18	ObjectType	word (Hi-Lo)
20	ObjectNameLen	byte
21	ObjectName	byte[ObjectNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	No Allocation Space
193	0xC1	No Account Balance
194	0xC2	Credit Limit Exceeded
197	0xC5	Server Login Locked
217	0xD9	Maximum Logins Exceeded
218	0xDA	Bad Login Time
219	0xDB	Node Address Violation
220	0xDC	Account Expired
222	0xDE	Bad Password

## See Also

**Get Login Key 0x2222 23 23 (page 195)**

# Login Object 0x2222 23 20

Enables a client to identify itself to the file server and gain rights to access certain directories (and files) within the file server.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (20)	byte
10	ObjectType	word (Hi-Lo)
12	ClientNameLen	byte
13	ClientName	byte[ClientNameLen]
13 + ClientNameLen	PasswordLen	byte
14 + ClientNameLen	Password	byte>PasswordLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
193	0xC1	No Account Balance
194	0xC2	Credit Limit Exceeded
197	0xC5	Server Login Locked
214	0xD6	Unencrypted Password
215	0xD7	Account Bad
217	0xD9	Maximum Logins Exceeded
218	0xDA	Bad Login Time
219	0xDB	Node Address Violation
222	0xDE	Bad Password
223	0xDF	Old Password

Decimal	Hex	Description
232	0xE8	Write To Group
236	0xEC	No Such Set
237	0xED	Property Exists
239	0xEF	Illegal Name
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
242	0xF2	No Object Read
246	0xF6	No Property Delete
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

Each server keeps a small database of information, called the bindery, about various named objects and their associated properties. File server "users" are one of the types of objects that appear in the bindery.

Most client objects identify themselves as Object Type 1, type USER. See “[ObjectType](#)” on [page 107](#) for other possible object types.

Receipt of a **Login Object** request causes the file server to automatically release any access privileges that the client previously had.

*SubFuncStrucLen = 5 + ClientNameLen + PasswordLen*

## See Also

[Login User \(old\) 0x2222 23 00 \(page 210\)](#), [Logout 0x2222 25 \(page 212\)](#), [Keyed Object Login 0x2222 23 24 \(page 207\)](#), [Get Login Key 0x2222 23 23 \(page 195\)](#)

# Login User (old) 0x2222 23 00

Logs in an object of type USER.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (0)	byte
10	UserNameLen	byte
11	UserName	byte[UserNameLen]
11 + UserNameLen	PasswordLen	byte
12 + UserNameLen	Password	byte[PasswordLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Space
193	0xC1	No Account Balance
194	0xC2	Credit Limit Exceeded
197	0xC5	Server Login Locked
215	0xD7	Account Bad
217	0xD9	Maximum Logins Exceeded
218	0xDA	Bad Login Time
219	0xDB	Node Address Violation
222	0xDE	Bad Password
223	0xDF	Old Password
232	0xE8	Write To Group
236	0xEC	No Such Set

Decimal	Hex	Description
237	0xED	Property Exists
239	0xEF	Illegal Name
240	0xF0	Illegal Wildcard
241	0xF1	Bindery Security
242	0xF2	No Object Read
246	0xF6	No Property Delete
251	0xFB	No Such Property
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

**Login User (old)** is still used in the NetWare 2.0a shell but is no longer called in NetWare 2.1 and above shells.

*SubFuncStrucLen = 3 + UserNameLen + PasswordLen*

## See Also

**Logout 0x2222 25 (page 212)**

# Logout 0x2222 25

Allows a client to relinquish its current server access privileges without breaking its service connection.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (25)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful



# Negotiate Buffer Size 0x2222 33

Allows a client to negotiate the buffer size that it will use when sending file read and write requests to the server.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (33)	byte
7	ProposedBufferSize	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	AcceptedBufferSize	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Clients are constrained by the buffer size in two ways. First, any file read or file write request cannot ask for more data than will fit in the negotiated buffer size. Second, any file read or file write requests can not cross a logical byte boundary that corresponds to the negotiated buffer size.

*ProposedBufferSize* contains the buffer size the client wants to use. This buffer size is only a proposed buffer size and can be modified by the server. *AcceptedBufferSize* contains the buffer size that the client and the server must actually use for file reading and writing. *AcceptedBufferSize* will be less than or equal to *ProposedBufferSize*.

Buffer sizes are constrained to one of the following sizes:

- .5K    512 bytes
- 1K    1,024 bytes
- 2K    2,048 bytes
- 4K    4,096 bytes
- 8K    8,192 bytes

16K 16,384 bytes

32K 32,768 bytes

When a service connection is first established, the server assumes a default buffer size of 512 bytes.

## See Also

**Read From a File 0x2222 72 (page 431), Write to a File 0x2222 73 (page 509)**

# Request Being Processed 0x9999

Signals the client that the server has received a duplicate service request and that the original request is still being processed.

**NetWare Server:** 3.x, 4.x, 5.x

## Reply Format

Offset	Content	Type
0	ReplyType (0x9999)	word (Hi-Lo)
2	SequenceNumber (N/A)	byte
3	ConnectionNumber (ConnectionNumber)	byte
4	TaskNumber (TaskNumber)	byte
5	Reserved (N/A)	byte
6	CompletionCode (N/A)	byte
7	ConnectionStatus (N/A)	byte

## Remarks

This message can be the response to any service request made by the client. If a client receives this request, it must reset its timeout and retry counters and continue to wait for the actual service reply to arrive from the server. If the service reply does not arrive before the timeout counter expires, the client should repeat the request.

If the server and the client use a guaranteed delivery message protocol (such as SPX) to communicate, this message does not need to be returned to the caller.

The following message is not a request message:

`This Request Being Processed`

## See Also

**Request Processed 0x3333 (page 216)**

# Request Processed 0x3333

Signals the client that the server has completed servicing a request.

**NetWare Server:** 3.x, 4.x, 5.x

## Reply Format

Offset	Content	Type
0	ReplyType (0x3333)	word (Hi-Lo)
2	SequenceNumber (RequestSequenceNumber)	byte
3	ConnectionNumber (ServiceConnection)	byte
4	TaskNumber (ClientTaskNumber)	byte
5	Reserved (0)	byte
6	CompletionCode	byte
7	ConnectionStatus (StatusFlags)	byte

## Remarks

The Reply Format is valid for all functions except **Create Service Connection** and **Destroy Service Connection**.

*SequenceNumber*, *ConnectionNumber*, and *TaskNumber* will match the request fields by the same name for the corresponding request.

*CompletionCode* should be checked for all replies; a nonzero value indicates that an error occurred while the client's request was being processed by the file server.

*ConnectionStatus* contains the following bits of information:

Bit 0 Set if the client's request contained a bad service connection

Bit 2 Set if the client requested a service connection and the file server has no further connections available

Bit 4 Set if the file server is down

Bit 6 Set if the file server is holding a broadcast message for the client

ConnectionStatus should be checked by the client each time a service response message ("Request Processed" or "Request Being Processed") is received. If Bit 4 is set, the client knows that the service connection has been terminated.

The following message is not a request message:

`This Request Processed`

## See Also

[Request Being Processed 0x9999 \(page 215\)](#)

# Set Watchdog Delay Interval 0x2222 23 30

Allows a connection to specify the watchdog delay interval.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFunctionLen (5)	word (Hi-Lo)
9	SubFunctionCode (30)	byte
10	NumberOfMinutesToDelay	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Number of Minutes to Delay

## Remarks

The watchdog delay interval is the time the server delays before the watchdog begins checking to see if the connection is still active. By setting the watchdog delay interval, a connection can power down for up to two weeks and still maintain its connection with the server. A connection can delay watchdog packets only for itself.

*NumberOfMinutesToDelay* can have the following values (in minutes):

0 Reset the connection to the default

1-20,199 Amount of time to delay the watchdog packets (one minute to approximately two weeks).

# Set Connection Language Encoding 0x2222 23 34

Allows you to set the code page of the connection to the same code page as the client who initiated the connection.

**NetWare Server:** 6.5 SP1 and later

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (32)	word (Hi-Lo)
9	SubFuncCode (34)	byte
10	SetMask	long (Lo-Hi)
14	NCPEncodedStringsBits	long (Lo-Hi)
18	CodePage	long (Lo-Hi)
22	Reserved	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Not Connected
253	0xFD	Bad Station Number

## Remarks

The client can set the UTF8\_NCP\_STRINGS\_ENABLED (0x1000) bit, which indicates to a supporting file system that NCP request path strings are UTF8 encoded, in the status field of the connection. For any volumes that support the UTF8 encoded feature, the VIUTF8NCPStringsEnabledBit (0x80) bit is set in the statusFlagBits field (which can be retrieved by calling [Get Extended Volume Information 0x2222 22 51](#) (page 345)).

To change the UTF8\_NCP\_STRINGS\_ENABLED status field in the connection, you must set the SetMask field to NCPEncodedStringsBits. To remove the UTF8\_NCP\_STRINGS\_ENABLED status field, you must set the SetMask field to NCPEncodedStringsBits and set the NCPEncodedStringsBits field to 0 before making the NCP request. The NCPEncodedStringsBits field and the CodePage field can both be set in the same request, provided that the proper SetMask bits are set.

The bit mask for the SetMask field follows:

Bit(s)	Definition
0	NCP Encoded Strings Bit
1	Connection Code Page Bit
2-15	reserved

## See Also

[Get Extended Volume Information 0x2222 22 51 \(page 345\)](#), [Enumerate Connection Information from Connection List 0x2222 123 16 \(page 978\)](#)



# 12 Structures

This section describes the Connection structures and their fields.

# NetworkAddressStruct

Returns the internet address information.

## Syntax

```
struct NetworkAddressStruct
{
    byte    NetworkAddress[4];
    byte    NetworkNodeAddress[6];
    word    NetworkSocket (Hi-Lo)
};
```

## Remarks

For an IP-only server, the 4-byte IP address is placed into the 4-byte *NetworkAddress* field of the *NetworkAddressStruct*, while the *NetworkNodeAddress* and *NetworkSocket* fields are left blank.

# VI

## Data Migration

Data Migration NCPs allow workstations or NLMs to do the following:

- ♦ Retrieve information about Data Migration files or a specific volume
- ♦ Retrieve Data Migration support module information or information about the Data Migration NLM
- ♦ Get or Set the default read-write support module ID
- ♦ Migrate or demigrate files to a device supported by a Data Migration support module

Data Migration is the process of migrating or moving data from a NetWare volume to an online data media such as disk, tape, or CD ROM. Although data is migrated, the NetWare file system continues to perceive data as residing on a NetWare volume. Data Migration offers NetWare users a way of optimizing space on a NetWare volume. For example, by migrating data in files that are least recently used or least frequently accessed, users gain optimum space on NetWare volumes, yet can see and, if necessary, retrieve migrated data with relative speed.

Because Data Migration is a real-time process, a user at a workstation can request data migration/demigration for a specified file, and the migration/demigration will occur immediately and transparently to the user. Users can also mark files as ineligible for migration.

To access Data Migration NCPs, use the following:

- ♦ [Chapter 13, “Concepts,” on page 225](#)
- ♦ [Chapter 14, “NCPs,” on page 227](#)
- ♦ [Chapter 15, “Structures,” on page 245](#)



# 13 Concepts

This section explains ideas that are common to Data Migration NCPs.

## Support Modules

Data Migration is implemented in NetWare 4.0 as the Data Migration NLM and as various, optional Data Migration Support Module NLMs. Up to 32 Support Modules and their associated devices, whether hard disk, tape, CD ROM, etc. can register with Data Migration.



# 14 NCPs

This section describes each of the Data Migration NCPs, their Request and Reply formats, and Return Values.

# DM File Information 0x2222 90 129

Returns information about Data Migration files.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (129)	byte
10	Volume	long (Lo-Hi)
14	DirectoryEntry	long (Lo-Hi)
18	NameSpace	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	SupportModuleID	long (Lo-Hi)
12	RestoreTime	long (Lo-Hi)
16	DMInfoEntries	long (Lo-Hi)
xx+	<b>DMINFO (page 246)</b>	structure

## Parameters

### *Volume*

(Request) Specifies the NetWare volume for which information is requested.

### *DirectoryEntry*

(Request) Specifies the directory entry for which information is requested.

### *NameSpace*

(Request) Specifies the name space of the file(s) for which information is requested.

### *SupportModuleID*

(Reply) Specifies the Support Module containing the migrated data.



### *RestoreTime*

Specifies the estimated time in seconds that the support module believes that it will take to demigrate the file.

### *DMInfoEntries*

(Reply) Specifies the number of valid DMINFO entries.

### *DMINFO*

(Reply) Points to the DMINFO structure, which contains information about the Data Migration data streams.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
152	0x98	Invalid Volume
168	0xA8	Access Denied

# DM Support Module Information 0x2222 90 132

Returns information about Data Migration NLM support modules or a list of all loaded support module IDs.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (132)	byte
10	InfoLevel	long (Lo-Hi)
14	SupportModuleID	long (Lo-Hi)

## Reply Format (InfoLevel = 0)

Offset	Content	Type
Reply header		
8	IOFlag	long (Lo-Hi)
12	SMInfoSize	long (Lo-Hi)
16	AvailableSpace	long (Lo-Hi)
20	UsedSpace	long (Lo-Hi)
24+	SModuleName	byte
xx+	SMInfo	byte

## Reply Format (InfoLevel = 1)

Offset	Content	Type
Reply header		
8	NumberOfSMs	long (Lo-Hi)
12+	SMIDs[]	long (Lo-Hi)

# Reply Format (InfoLevel = 2)

Offset	Content	Type
Reply header		
8	NameLen	byte
9+	Name	byte[]

## Parameters

- InfoLevel*  
(Request) Specifies the type of information to retrieve.
- SupportModuleID*  
(Request) Specifies the ID number of the support module.
- IOFlag*  
(Reply) Specifies the read and write access status.
- SMInfoSize*  
(Reply) Specifies the size of the specific device information block (maximum 384 bytes).
- AvailableSpace*  
(Reply) Specifies the amount of space left free on the support module.
- UsedSpace*  
(Reply) Specifies the amount of space used through the support module.
- SModuleName*  
(Reply) Specifies the name of the support module.
- SMInfo*  
(Reply) Specifies the information about a selected Data Migration support module.
- NumberOfSMs*  
(Reply) Specifies the number of valid support module IDs.
- SMIDs*  
(Reply) Specifies the list of support module IDs.
- NameLen*  
Specifies the length of the support module name.
- Name*  
Specifies the name of the support module that has been loaded and registered.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
168	0xA8	Invalid Support Module ID
255	0xFF	Failure or Invalid Info Level

# DM Support Module Capacity Request 0x2222 90 135

Queries the server for the support module information, which includes the block size in sectors, total blocks, and used blocks.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (1+ SMCapacityReq)	word (Hi-Lo)
9	SubFuncCode (135)	byte
10	CapacityRequest	SMCapacityReq (page 247)

## Reply Format

Offset	Content	Type
Reply header		
8	SMBlockSizeInSectors	long (Lo-Hi)
12	SMTotalBlocks (-1 = unlimited)	long (Lo-Hi)
16	SMUsedBlocks	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
168	0xA8	Invalid Support Module ID

# Get/Set Default Read-Write Support Module ID 0x2222 90 134

Returns or sets the default support module ID.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (134)	byte
10	Get/Set Flag	long (Lo-Hi)
14	SupportModuleID	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	SupportModuleID	long (Lo-Hi)

## Parameters

### *Get/SetFlag*

(Request) Specifies the default read/write support module ID to set or return:

- 0 Returns the default support module ID
- 1 Sets the default support module ID

### *SupportModuleID*

(Request) Specifies whether to clear or return the default support module ID:

- NULL (Set) Clear the default support module ID
- NULL (Get) No default support module ID has been set

### *SupportModuleID*

(Reply) Specifies the ID of the support module.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
236	0xEC	Invalid Support Module ID
251	0xFB	No Such Property or Invalid Get/Set Flag

# Migrator Status Info 0x2222 90 131

Returns information about the migrator.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (131)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	DMPresentFlag	long (Lo-Hi)
12	DMmajorVersion	long (Lo-Hi)
16	DMminorVersion	long (Lo-Hi)

## Parameters

### *DMPresentFlag*

(Reply) Specifies whether the Data Migration NLM has been loaded:

0 Data Migration NLM is not loaded

1 Data Migration NLM has been loaded and is running

### *DMmajorVersion*

(Reply) Specifies the major version number of the migrator.

### *DMminorVersion*

(Reply) Specifies the minor version number of the migrator.

## Return Values

Decimal	Hex	Description
0	0x00	Successful



Decimal	Hex	Description
126	0x7E	Invalid Length

# Move File Data From DM 0x2222 90 133

Moves a file back to the specified NetWare volume.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (133)	byte
10	Volume	long (Lo-Hi)
14	DirectoryEntry	long (Lo-Hi)
18	NameSpace	long (Lo-Hi)

## Parameters

*Volume*

(Request) Specifies the NetWare volume to be demigrated.

*DirectoryEntry*

(Request) Specifies the directory entry to be demigrated.

*NameSpace*

(Request) Specifies the name space associated with the information to be demigrated.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
120	0x78	Service Unavailable on this Volume
126	0x7E	Invalid Length
152	0x98	Disk Map Error (Invalid Volume)
156	0x9C	Invalid Directory Entry
168	0xA8	Access Denied

## Remarks

In NetWare 4.10, an invalid *nameSpace* parameter will abend the server.

# Move File Data To DM 0x2222 90 128

Moves a file's data to some online long-term storage media but leaves the file visible on a NetWare volume.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (21)	word (Hi-Lo)
9	SubFuncCode (128)	byte
10	Volume	long (Lo-Hi)
14	DirectoryEntry	long (Lo-Hi)
18	NameSpace	long (Lo-Hi)
22	SupportModuleID	long (Lo-Hi)
26	Flags	long (Lo-Hi)

## Parameters

### *Volume*

(Request) Specifies the NetWare volume to be migrated.

### *DirectoryEntry*

(Request) Specifies the directory entry to be migrated.

### *NameSpace*

(Request) Specifies the name space associated with the information to be migrated.

### *SupportModuleID*

(Request) Specifies the assigned ID number of the support module to migrate data to or specifies the default support module value:

```
#define UseDefaultRdWrSupportModule 0x64646d6d
```

### *Flags*

(Request) Specifies the type of media the information will be migrated to (0 or 1).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
152	0x98	Disk Map Error
156	0x9C	Invalid Path
168	0xA8	Invalid Support Module ID

## Remarks

In NetWare 4.10, an invalid *nameSpace* parameter will abend the server.

# RTDM Request 0x2222 90 136

NetWare Server: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFuncCode (136)	byte
10	Verb	long (Lo-Hi)
14	VerbData	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Volume DM Status 0x2222 90 130

Returns information about the Data Migration NLM on a specified volume.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (130)	byte
10	Volume	long (Lo-Hi)
14	SupportModuleID	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NumOfFilesMigrated	long (Lo-Hi)
12	TtlMigratedSize	long (Lo-Hi)
16	SpaceUsed	long (Lo-Hi)
20	LimboUsed	long (Lo-Hi)
24	SpaceMigrated	long (Lo-Hi)
28	FileLimbo	long (Lo-Hi)

## Parameters

*Volume*

(Request) Specifies the NetWare volume for which status information is requested.

*SupportModuleID*

(Request) Specifies the ID number of the support module for which information is requested.

*NumOfFileMigrated*

(Reply) Specifies the number of files from the selected volume that has been migrated.

*TtlMigratedSize*

(Reply) Specifies the total size needed to recover all the data on the selected volume.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
120	0x78	Service Unavailable on this Volume
126	0x7E	Invalid Length
152	0x98	Disk Map Error



# 15 Structures

This section describes the Data Migration structures and their fields.

# DMINFO

Returns information about the Data Migration data streams.

## Syntax

```
typedef struct
{
    LONG    DataSize;
} DMINFO;
```

# SMCapacityReq

Specifies information about the support module for which to return information.

## Syntax

```
typedef struct
{
    LONG    SupModID;
    LONG    Vol;
    LONG    TargetDirectoryBase;
} SMCapacityReq;
```



# VII

## Extended Attributes

Extended Attributes NCPs allow you to close, enumerate, duplicate, read, and write a file's extended attributes.

To access Extended Attributes NCPs, use the following:

- ♦ [Chapter 16, “Concepts,” on page 251](#)
- ♦ [Chapter 17, “NCPs,” on page 253](#)
- ♦ [Chapter 18, “Structures,” on page 265](#)
- ♦ [Chapter 19, “Values,” on page 271](#)



# 16 Concepts

This section explains ideas that are common to Extended Attributes NCPs.

## Read and Write Positions

All read and write positions are based on a 128-byte page in the file server; therefore, read and write positions always start on a mod(128)-byte boundary. All data is passed between the server and client in 512-byte data chunks, except that the last transfer can be less than 512-bytes.

Extended Attribute values are always read or written completely; no partial read or writes are allowed.

The starting value of the read or write position is zero. Some Extended Attribute NCPs return the next read or write position to use for the next request.





# 17

## NCPs

This section describes each of the Extended Attributes NCPs, their Request and Reply formats, and Return Values.

# Close Extended Attribute Handle 0x2222 86 01

Closes an extended attribute handle.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (86)	byte
7	SubFunction (01)	byte
8	reserved (0)	word (Lo-Hi)
10	ECHandle	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	SUCCESSFUL
207	0xCF	ERR_INVALID_EA_HANDLE
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED

# Duplicate Extended Attributes 0x2222 86 05

NetWare Server: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (86)	byte
7	SubFunction (05)	byte
8	SrcFlags (see “Flags Values” on page 272)	word (Lo-Hi)
10	DstFlags (see “Flags Values” on page 272)	word (Lo-Hi)
12	SrcEAHandleStruct (page 266)	structure
20	DstEAHandleStruct (page 266)	structure

## Reply Format

Offset	Content	Type
Reply header		
8	DuplicateCount	long (Lo-Hi)
12	DataSizeDuplicated	long (Lo-Hi)
16	KeySizeDuplicated	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	SUCCESSFUL
209	0xD1	ERR_EA_ACCESS_DENIED
		ERR_NO_SET_PRIVILEGES
		ERR_NO_ALLOC_SPACE
		ERR_EA_INTERNAL_FAILURE

## Remarks

You cannot use the EA Handle or Immediate Close Handle settings in the flags parameters.

# Enumerate Extended Attribute 0x2222 86 04

NetWare Server: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (86)	byte
7	SubFunction (04)	byte
8	Flags (see “Flags Values” on page 272)	word (Lo-Hi)
10	EAHandleStruct (page 266)	structure
18	InspectSize	long (Lo-Hi)
22	EnumerateSequence	word (Lo-Hi)
24	KeyLength	word (Lo-Hi)
26	Key[]	byte

## Reply Format (InformationLevel = 0)

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	reserved (0)	word (Lo-Hi)
30	reserved (0)	word (Lo-Hi)

## Reply Format (InformationLevel = 1)

Offset	Content	Type
Reply header		

Offset	Content	Type
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	NextEnumerateSequence	word (Lo-Hi)
30	EnumEAStructCount	word (Lo-Hi)
32	EnumEAStruct_Lvl1 (page 267)	structures

## Reply Format (InformationLevel = 6)

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	reserved (0)	word (Lo-Hi)
30	reserved (0)	word (Lo-Hi)
32	EnumEAStruct_Lvl6 (page 268)[]	structure

## Reply Format (InformationLevel = 7)

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)

Offset	Content	Type
28	NextEnumerateSequence	word (Lo-Hi)
30	EnumEAStructCount	word (Lo-Hi)
32	EnumEAStruct_Lvl7 (page 269)[]	structure

## Return Values

Decimal	Hex	Description
0	0x00	SUCCESSFUL
201	0xC9	ERR_EA_NOT_FOUND
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
		ERR_INTERNAL_FAILURE
		ERR_NO_ALLOC_SPACE
		ERR_UNKNOWN_REQUEST

## Remarks

The first time **Enumerate Extended Attribute** is called, *EnumerateSequence* must be zero. For subsequent requests, set *EnumerateSequence* to the value previously returned in *NextEnumerateSequence*.

If KeyLength equals zero, a list of all Extended Attributes, including the information about each Extended Attribute, is returned. If a key is used, only information Level 6 may be used.

When you are using a Level 6 enumerate request, *TtlEAsDataSize* and *TtlEAsKeySize* contain the size of only the specified value and key, while *TtlEAs* contains one. *KeyPages* and *ValuePages* in the structure contain the number of Extended Directory pages that were used.

See “**Flags Values**” on page 272 for the description of *InformationLevel*.

# Read Extended Attribute 0x2222 86 03

NetWare Server: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (86)	byte
7	SubFunction (03)	byte
8	Flags (see “Flags Values” on page 272)	word (Lo-Hi)
10	EAHandleStruct (page 266)	structure
18	ReadPosition	long (Lo-Hi)
22	InspectSize	long (Lo-Hi)
26	KeyLength	word (Lo-Hi)
28	Key[]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlValuesLength	long (Lo-Hi)
16	NewEAHandle	long (Lo-Hi)
20	AccessFlag (see “Access Flag Values” on page 271)	long (Lo-Hi)
24	ValueLength	word (Lo-Hi)
26	Value[]	byte

## Return Values

Decimal	Hex	Description
0	0x00	SUCCESSFUL



Decimal	Hex	Description
201	0xC9	ERR_EA_NOT_FOUND
206	0xCE	ERR_EA_BAD_DIR_NUM
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED
		ERR_NO_SET_PRIVILEGES
		ERR_NO_ALLOC_SPACE
		ERR_EA_INTERNAL_FAILURE
		ERR_EA_INSPECT_FAILURE
		ERR_INVALID_FILE_HANDLE
		ERR_INVALID_PATH

## Remarks

*Key* should be sent only on the first request; subsequent requests should not contain a key value. Set *KeyLength* to zero after the first request.

# Write Extended Attribute 0x2222 86 02

NetWare Server: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (86)	byte
7	SubFunction (02)	byte
8	Flags (see “Flags Values” on page 272)	word (Lo-Hi)
10	EAHandleStruct (page 266)	structure
18	TtlWriteDataSize	long (Lo-Hi)
22	WritePosition	long (Lo-Hi)
26	AccessFlag (see “Access Flag Values” on page 271)	long (Lo-Hi)
30	ValueLength	word (Lo-Hi)
32	KeyLength	word (Lo-Hi)
34	Key[]	byte
xx	Value[]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	BytesWritten	long (Lo-Hi)
16	NewEAHandle	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	SUCCESSFUL

Decimal	Hex	Description
200	0xC8	ERR_MISSING_EA_KEY
201	0xC9	ERR_EA_NOT_FOUND
203	0xCB	ERR_EA_NO_KEY_NO_DATA
206	0xCE	ERR_EA_BAD_DIR_NUM
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
210	0xD2	ERR_DATA_PAGE_ODD_SIZE
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED
212	0xD4	ERR_BAD_PAGE_BOUNDARY
		ERR_NO_SET_PRIVILEGES
		ERR_EA_INTERNAL_FAILURE
		ERR_INVALID_PATH
		ERR_EA_WRITE_OUT_OF_RANGE
		ERR_INSUFFICIENT_SPACE
		ERR_HARD_FAILURE

## Remarks

*Key* should be sent only on the first request; subsequent requests should not contain a key value. Set *KeyLength* to zero after the first request.



# 18 Structures

This section describes the Extended Attributes structures and their fields.

# EAHandleStruct

## Bit Definitions

Bits 1 and 0 of the flags definition are reserved for EAHandleStruct as follows:

Bit 1	Bit 0	EA Handle Structure
0	0	LONG EAHandle;LONG reserved;
0	1	BYTE NetWareHandle[4];LONG reserved;
1	0	LONG Volume;LONG DirectoryNumber;
1	1	reserved

# EnumEAStruct\_Lvl1

## Syntax

Offset	Content	Type
0	ValueLength (variable)	LONG (Lo-Hi)
4	KeyLength (variable)	WORD (Lo-Hi)
6	AccessFlag (variable)	LONG (Lo-Hi)
10	Key[] (variable)	BYTE

# EnumEAStruct\_Lvl6

## Syntax

Offset	Content	Type
0	ValueLength (variable)	LONG (Lo-Hi)
4	KeyLength (variable)	WORD (Lo-Hi)
6	AccessFlag (variable)	LONG (Lo-Hi)
10	KeyExtants (variable)	LONG (Lo-Hi)
14	ValueExtants (variable)	LONG (Lo-Hi)
18	Key[] (variable)	BYTE



# EnumEAStruct\_Lvl7

## Syntax

Offset	Content	Type
0	KeyLength (variable)	BYTE
1	Key[] (variable)	BYTE
x	NullByte (0)	BYTE



# 19 Values

This section describes the common values used by Extended Attributes NCPs.

## Access Flag Values

AccessFlag is defined as follows:

- ♦ Bits 31-16 are client-definable flags
- ♦ Bits 15-0 are defined as:

Bits	Definition
15-8	reserved
7	EA Need Bit Flag
6-0	reserved

## EA Return Values

The server can return the following values:

Decimal	Hex	Constant
0	0x00	SUCCESSFUL
200	0xC8	ERR_MISSING_EA_KEY
201	0xC9	ERR_EA_NOT_FOUND
202	0xCA	ERR_INVALID_EA_HANDLE_TYPE
203	0xCB	ERR_EA_NO_KEY_NO_DATA
204	0xCC	ERR_EA_NUMBER_MISMATCH
205	0xCD	ERR_EXTENT_NUMBER_OUT_OF_RANGE
206	0xCE	ERR_EA_BAD_DIR_NUM
207	0xCF	ERR_INVALID_EA_HANDLE
208	0xD0	ERR_EA_POSITION_OUT_OF_RANGE
209	0xD1	ERR_EA_ACCESS_DENIED

Decimal	Hex	Constant
210	0xD2	ERR_DATA_PAGE_ODD_SIZE
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED
212	0xD4	ERR_BAD_PAGE_BOUNDARY
213	0xD5	ERR_INSPECT_FAILURE
214	0xD6	ERR_EA_ALREADY_CLAIMED
215	0xD7	ERR_ODD_BUFFER_SIZE
216	0xD8	ERR_NO_SCORECARDS
217	0xD9	ERR_BAD_EDS_SIGNATURE
218	0xDA	ERR_EA_SPACE_LIMIT
219	0xDB	ERR_EA_KEY_CORRUPT
220	0xDC	ERR_EA_KEY_LIMIT
221	0xDD	ERR_TALLY_CORRUPT

## Flags Values

Bit 7 is defined as follows:

Bit 7	Definition
0	
1	Immediate Close Handle

Only levels 0, 1, and 7 are currently defined for *InformationLevel*. Bits 6-4 are reserved for the *InformationLevel* used only by **Enumerate Extended Attribute** as follows:

Bit 6	Bit 5	Bit 4	Information Level
0	0	0	Level 0
0	0	1	Level 1
0	1	0	Level 2
0	1	1	Level 3
1	0	0	Level 4
1	0	1	Level 5
1	1	0	Level 6
1	1	1	Level 7

Bit 2 is defined as follows:

Bit 2	Definition
0	
1	Close Handle on Error (Bit 7 = NZ)

Bits 1 and 0 of the flags parameters definition are reserved for the [EAHandleStruct](#).

## Multiple Purpose NCP Return Values

**Enumerate Extended Attribute**, **Read Extended Attribute**, and **Write Extended Attribute** can perform multiple actions, such as opening an Extended Attribute handle, performing the appropriate function, and closing the Extended Attribute handle.

The Return Value field is defined as 32 bits, with the lower order 16 bits containing the error code from the server. In addition to this server error code, the upper 16 bits indicate which multiple action failed as follows:

Bit Number					Definition
31	30	29	...	16	
x					Enumerate Failed
	x				Read Failed
		x			Write Failed



# VIII File System

File System NCPs enable you to perform tasks that fall into two broad categories:

First, you can obtain information about files, volumes, and directories. For example, you can get the following types of information: a file's size, file or subdirectory information, directory rights, name space information, a volume name or number, etc.

Second, you can manipulate files and directories. For example, you can create, open, read, write, close, delete, and rename files. You can also manipulate extended file attributes, restore erased files, purge deleted files, set and scan file information, and copy files between directories on the same file server.

By using directory-based NCPs, you can create, rename, and delete directories. You can also modify a directory's maximum rights mask, add and delete directory trustees, allocate and deallocate directory handles, etc.

To access File System NCPs, use the following:

- ♦ [Chapter 20, "Concepts," on page 277](#)
- ♦ [Chapter 21, "NCPs," on page 285](#)
- ♦ [Chapter 22, "Enhanced NCPs," on page 511](#)
- ♦ [Chapter 23, "Structures," on page 569](#)
- ♦ [Chapter 24, "Values," on page 633](#)





# 20 Concepts

This section explains ideas that are common to File System NCPs.

## File Security

The server strictly enforces file access security to ensure that clients are allowed access to files for which they have been granted rights only. Also, the server coordinates file access among clients so that file data integrity can be maintained in the distributed multi-user environment created by multiple clients contending for the same information.

A client's access to files in a directory area is controlled by the effective rights the client has in the directory. These rights are specified in Directory Access Rights.

## File Attributes

Each file has four file attributes (bytes 0-3) that control access to that file. The bits of each of the four file attribute bytes are defined as follows:

---

### Byte 0 Bit Definitions

Bit	Name	Description
0	Read Only	File can be read but not written to.
1	Hidden	File will not be shown in a normal directory listing.
2	System	File will not be shown in a normal directory listing.
3	Execute Only	File can be loaded for execution only; once this bit has been set, it cannot be cleared.
4	Subdirectory	Entry is a subdirectory, not a file.
5	Archive	The file has been changed since it was last backed up.
6	Execute Confirm	
7	Shareable	The file can be simultaneously accessed by more than one user.

---

---

### Byte 1 Bit Definitions

Bit	Name	Description
8-10	Search Mode	These first three bits are valid only with NetWare 2.0a and above.

---

---

**Byte 1 Bit Definitions**

Bit	Name	Description
11	Indexed	If set, the OS will index the file's sectors in the File Allocation Tables (FATs), which reduces the time it takes to access the file.
12	Transactional	If set, TTS will track all writes to the file during a transaction. A transaction file cannot be deleted or renamed.
13	Reserved	
14	Read Audit	This bit can be set by users with supervisor security equivalence only.
15	Write Audit	This bit can be set by users with supervisor security equivalence only.

---

---

**Byte 2 Bit Definitions**

Bit	Name
16	Immediate Purge Bit
17	Rename Inhibit Bit
18	Delete Inhibit Bit
19	Copy Inhibit Bit
20	File Auditing Bit
21	Reserved
22	Data Migrated Bit
23	Inhibit Data Migration Bit

---

---

**Byte 3 Bit Definitions**

Bit	Name
24	Data Migration Save Key Bit
25	Compress Immediate Bit
26	Data Stream Compressed Bit
27	Compression Inhibit Bit
28	Reserved
29	Can't Compress Data Bit
30	Attributes Archive Bit
31	Reserved

---

Search Mode can have the following values:

Bit Offset			Decimal Equivalent	Search Mode
10	9	8		
0	0	0	0	Search On All Read Only Opens
0	0	1	1	Search On Read Only Opens With No Path
0	1	0	2	Shell Default Search Mode
0	1	1	3	Search On All Opens With No Path
1	0	0	4	Do Not Search
1	0	1	5	Reserved
1	1	0	6	Search On All Opens
1	1	1	7	Reserved

Clients can access files by name when files are being created, deleted, renamed, searched for, and opened. When a file is opened, the server gives the client a 6-byte file handle that the client must supply to the server when the client accesses the file for reading, writing, or closing.

To a client, a file is a logical byte stream. The client can read or write records of varying sizes starting at any byte offset within the file.

A client must specify the search attributes when making requests to open a file, rename a file, erase a file, search for a file, set a file's attribute byte, or get/set a file's extended attribute byte. The bits of the search attributes byte correspond to the bits in the file attributes byte. However, the bits that are significant in the search attribute byte are only the hidden and system bits. A normal file is a file whose system and hidden bits are clear in its file attribute byte.

The following table indicates the search attributes that are recognized by the file server:

Hidden Bit	System Bit	Recognized by File Server
0	0	Normal files only
0	1	Normal files and system files only
1	0	Normal files and hidden files only
1	1	All files

## File Names

In each function that requires a file name, a client must provide a directory handle and a path and file name. The path and file name can contain directories down to any level, but the last part of the string must be the name of the desired file. The specified path is interpreted as a path that begins at the directory pointed to by the client's directory handle. If a client desires, a full path and file name, in the format "<F29>volume:directory/./directory/filename<F9>", and a directory handle of zero can be used.

File names can be up to 14 characters in length. They can contain any characters greater than 0x20 (ASCII space), with the following exceptions:

- ◆ " Double quote
- ◆ + Plus sign
- ◆ , Comma
- ◆ / Slash
- ◆ : Colon
- ◆ ; Semicolon
- ◆ < Less than sign
- ◆ = Equal sign
- ◆ > Greater than sign
- ◆ ? Question mark
- ◆ [ Left brace
- ◆ ] Right brace
- ◆ \ Backslash
- ◆ | Vertical bar

Periods are legal in a file name and are treated as any other legal character.

## Directory Access Rights

Several protocols have an Access Rights Mask field, which is used for either retrieving or setting directory security information.

For example, disk space on a file server is divided into one or more named logical volumes, with each volume having its own separate hierarchical directory structure. Each directory has a list of trustees and trustee access rights. A trustee in the trustee list is represented by the object ID of some object (user or group) that is given certain access privileges to the directory. A client's trustee number (also known as the Object ID) is determined when the client calls **Login Object** (0x2222 23 20).

Trustee access privileges are represented by a 1-byte mask with the following defined bits:

Bit	Function
0	If set, the trustee can read from files in this directory.
1	If set, the trustee can write to files in this directory.
2	If set, the trustee can open existing files in this directory.
3	If set, the trustee can create files in this directory.
4	If set, the trustee can delete files in this directory.
5	If set, the trustee has parental rights in this directory. Parental rights include the ability to create and delete subdirectories and the right to make other objects trustees of this directory or its subdirectories.
6	If set, the trustee can search this directory.

Bit	Function
7	If set, the trustee can modify the status flags of files in this directory and rename files.

The access rights granted to an object in a directory within a directory tree are inherited by that object in all that directory's subdirectories. The inherited rights are effective down only to a subdirectory in which other access rights have been granted.

In addition to the individual rights granted to each trustee, each directory has a maximum access rights mask, which indicates the access actions that can occur in the directory. A trustee is allowed the rights that appear in both the trustee's access rights mask and the directory's maximum access rights mask only (logical AND of the trustee's access right mask and the directory's maximum access rights mask).

The NetWare implementation of the file service protocols allows a client to claim security equivalence to as many as 33 trustees (user or group objects) at once. If object A is made security equivalent to object B, object B's object ID is added to the SECURITY\_EQUALS property of object A.

An object that is security equivalent to another object has the access rights of both objects combined. For example, object JOE (user) might be security equivalent to one or more other user or group objects. If JOE is a trustee in a directory and JOE is given read, open, and search access rights in that directory, JOE can search for, open, and read files in that directory and in all its subdirectories (if the maximum access rights of the directory and subdirectories include read, open, and search).

Later, JOE is made security equivalent to the SUPERVISOR object. JOE now has all rights of JOE as a trustee and JOE as a security equivalent of SUPERVISOR. Therefore, JOE can do anything that the object JOE is granted as a trustee of the directory and anything that the SUPERVISOR object can do in the directory. JOE's actual access rights in a given directory are computed by combining (logically ORing) JOE's access rights with the rights of the other trustees to which JOE is security equivalent and then logically ANDing these rights with the directory's maximum access rights mask.

## Directory Handle Table

Both the NetWare file server and the workstation maintain several tables used for directory services. A particular implementation must comply with the following design if it is to remain compatible with NetWare 2.x.

Each workstation attached to a given file server can allocate up to 256 directory handles on that file server. A directory handle is a number (00h to FFh) that points to a volume or a directory path. A file server maintains a Directory Handle Table for each attached workstation, which consists of 256 slots with each slot representing a directory handle. For example, the first slot represents directory handle 1, the second slot represents directory handle 2, etc. Each slot contains a volume name or a directory path. For example, slot 5 represents directory handle 05h, which points to the volume or directory path contained in slot 5.

You can call directory system functions to access up to 256 directory handles for the workstation on which an application is running. Once a directory handle is set, the directory handle can be used to specify a volume or a directory path on a file server.

Frequently, when you call a directory system function, you must target a particular directory in the request buffer by using one of the following three methods:

- ◆ Allocate a directory handle to point to the target directory and then specify the allocated directory handle in the request buffer.
- ◆ Specify both a directory handle and a complementary directory path in the request buffer. The directory handle must lead part of the way to the target directory, and the directory path must lead the rest of the way.
- ◆ Specify the entire directory path in the request buffer.

Directory system functions do not specify the file server since volume names and directory paths do not include file server names. Before calling a directory system function, you should ensure that the target file server is the current file server.

## Workstation Tables

A workstation maintains three tables that correspond to each other and relate to the file server's Directory Handle Table: a Drive Handle Table, a Drive Flag Table, and a Drive Connection ID Table. Directory system functions that allocate and deallocate directory handles map (or assign) workstation drives to directory handle by accessing these three workstation tables.

Each table consists of 32 1-byte entries. The first 26 slots of each table represent the permanent drive letters A to Z. The last 6 slots represent the temporary drive letters in the following order:

- ◆ [ Left bracket
- ◆ ] Right bracket
- ◆ \ Slash
- ◆ ^ Caret
- ◆ \_ Underscore
- ◆ ' Single quote

Each slot in the Drive Handle Table contains a directory handle number. For example, if slot 6 in the table contains 02h, drive letter F (the sixth letter) points to the volume or directory path contained in slot 2 of the file server's corresponding Directory Handle Table.

Each slot in the Drive Flag Table contains one of the following values:

---

0x00	The drive is unallocated.
0x01	The drive is a permanent network drive.
0x02	The drive is a temporary network drive.
0x80	The drive is a local drive.
0x81	The drive is a local drive allocated as a permanent network drive.
0x82	The drive is a local drive allocated as a temporary network drive.

---

Each slot in the Drive Connection ID Table contains a value (00h to 08h) identifying the server to which the drive letter is mapped. The value represents the position (slot 1 to slot 8) of the file server in the workstation's Server Name Table and Connection ID Table.

An application can allocate only the first 26 drive letters (A to Z) as permanent network drives. An application can allocate any of the 32 drive letters as temporary drives. A temporary drive remains allocated as long as the application remains active only.

## Directory Tables

To record information about directories, a file server maintains a Directory Table. Directory system functions that return directory information and manipulate rights access the Directory Table. The Directory Table consists of several kinds of 32-byte entries, including directory nodes, file nodes, and trustee nodes.

A directory node includes the following information about a directory: directory name, attribute byte, maximum rights mask, creation date, creator's object ID, a link to the parent directory, and a link to a trustee node (if one exists).

A file node includes the following information about a file: file name, attribute byte, extended attribute byte, file size, creation date, last accessed date, last updated date and time, and a link to a directory.

A trustee node includes the following information: the object IDs of one to five trustees of a directory that are linked to the trustee node, one to five corresponding trustee rights masks, a link to a directory, and a link to the next trustee node (if one exists).

## Volume Tables

To record information about volumes, a file server maintains a Volume Table. System functions that return information about volumes access the Volume Table. The Volume Table includes the number of volumes mounted in the file server, the names and sizes of each volume, and other information pertaining to each volume.

Scanning directories is accomplished by calling **File Search Initialize** (0x2222 62) and **File Search Continue** (0x2222 63).

## NetWare 3.1 and Above

NetWare 3.1 and above File System NCPs enable client-server communication. These NCPs are provided to allow the NetWare 3.1 and above OS to become independent of the client workstation, no matter which OS is running on the workstation. Name Space modules are still required, but internal hooks for each client are not required in the network OS.

Typically, clients request information from the server by setting selected bits or by filling fields of a particular structure with information. The server responds by returning information in the fields of a particular structure.

## Name Space Information Bit Mask

NSInfoBitMask is used to get name space information from or set name space information to a requested name space module. You determine which bits are valid by calling **Query NS Information Format 0x2222 87 23** (page 427) and passing in a requested volume and name space.

You can request the information you need by setting the appropriate bits to 1. If you don't need a specified piece of information, set the corresponding bit to zero.

Bit zero is defined to be a fixed-length field, which contains the file name (length preceded).



# 21

## NCPs

This section describes each of the File System Extension NCPs, their Request and Reply formats, and Return Values.

# Add Extended Trustee to Directory or File 0x2222 22 39

Adds a trustee and sets the trustee rights for a file or directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (8 + PathName)	word (Hi-Lo)
9	SubFunctionCode (39)	byte
10	DirHandle	byte
11	ObjectID	long (Hi-Lo)
15	TrusteeRights	word
17	PathLen	byte
18	Path	byte[PathLen]

## See Also

**Add Trustee to Directory 0x2222 22 13 (page 287), Delete Trustee from Directory 0x2222 22 14 (page 314)**

# Add Trustee to Directory 0x2222 22 13

Adds a new trustee to a directory's trustee list.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (8 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (13)	byte
10	DirectoryHandle	byte
11	TrusteeID	long (Hi-Lo)
15	TrusteeAccessMask	byte
16	DirectoryPathLen	byte
17	DirectoryPath	byte[DirectoryPathLen]

## Parameters

### *TrusteeID*

(Request) Specifies the valid ID of an object in the server's bindery; usually this number is retrieved from the server by calling the bindery functions.

### *TrusteeAccessMask*

(Request) Specifies the access rights to be given to the specified object for the target directory.

### *DirectoryPath*

(Request) Specifies the path relative to the specified directory handle.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error

Decimal	Hex	Description
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
252	0xFC	No Such Object
253	0xFD	Bad Station Number
255	0xFF	Hard Failure, Failure

## Remarks

**Add Trustee to Directory** replaces the access mask of the list trustee if the trustee already appears in the directory's trustee list; otherwise, the trustee is added to the directory's trustee list.

*DirectoryPath* must follow the same conventions as NetWare file names.

Only a client with access control rights to either the target directory its parent directory can call **Add Trustee to Directory**.

## See Also

**Add Extended Trustee to Directory or File 0x2222 22 39 (page 286)**, **Delete Trustee from Directory 0x2222 22 14 (page 314)**

# Add Trustee Set to File or Subdirectory 0x2222 87 10

NetWare Servers: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (10)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	SearchAttributes	word (Lo-Hi)
12	TrusteeRightsMask	word (Lo-Hi)
14	ObjectIDCount	word (Lo-Hi)
16	NWHandlePathStruct	structure
xx	TrusteeStruct[]	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If *TrusteeRightsMask* is set to -1, the individual *TrusteeRights* should be set to the appropriate rights for each entry in the *TrusteeStruct*.

*NWHandlePathStruct* is of type **NetWareHandlePathStruct** (page 609) and totals 307 bytes in length. *TrusteeStruct* starts after 307 bytes.

# Add User Disk Space Restriction 0x2222 22 33

Sets an object's volume disk space restriction.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (10)	word (Hi-Lo)
9	SubFunctionCode (33)	byte
10	VolumeNumber	byte
11	ObjectID	long (Hi-Lo)
15	DiskSpaceLimit	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error

## Remarks

All restrictions are in 4 KB blocks. Valid restrictions are 0 to 0x40000000.

## See Also

**Get Directory Disk Space Restriction 0x2222 22 35 (page 332), Remove User Disk Space Restrictions 0x2222 22 34 (page 438), Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366), Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476), Set Directory Disk Space Restriction 0x2222 22 36 (page 487), Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Alloc Permanent Directory Handle 0x2222 22 18

Creates a new permanent directory handle for the client and points the handle to the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (18)	byte
10	SourceDirectoryHandle	byte
11	HandleName	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryHandle	byte
9	AccessRightsMask	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
156	0x9C	Invalid Path
157	0x9D	No Directory Handles

Decimal	Hex	Description
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

*NewDirectoryHandle* is returned to the client together with the client's effective *AccessRightMask* in the specified directory. The client can specify the complete *DirectoryPath* name and give a *SourceDirectoryHandle* of zero.

The client must assign unique names to each directory handle it creates. If the client creates a permanent directory handle with the same name as a previously created permanent directory handle, the file server automatically deallocates the previous permanent directory handle.

Permanent directory handles outlive the client processes that created them. Such handles are maintained by the file server until they are overwritten by calling **Allocate Permanent Directory Handle** with the same handle name, until they are explicitly deleted by calling **Deallocate Directory Handle 0x2222 22 20**, or until the client releases its service connection privilege (logs out or destroys its service connection).

A client can have up to 255 directory handles in any combination of permanent and temporary handles.

## See Also

**Alloc Temporary Directory Handle 0x2222 22 19 (page 295), Alloc Special Temporary Directory Handle 0x2222 22 22 (page 293), Deallocate Directory Handle 0x2222 22 20 (page 310), Set Directory Handle 0x2222 22 00 (page 491)**



# Alloc Special Temporary Directory Handle 0x2222 22 22

Creates a temporary directory handle that cannot be remapped to another place.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (22)	byte
10	SourceDirectoryHandle	byte
11	HandleName	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryHandle	byte
9	AccessRightsMask	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
156	0x9C	Invalid Path
157	0x9D	No Directory Handles
161	0xA1	Directory I/O Error

Decimal	Hex	Description
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

**Alloc Special Temporary Directory** is functionally similar to **Alloc Temporary Directory Handle** (0x2222 22 19), except that the handle cannot be remapped by calling **Set Directory Handle** (0x2222 22 0).

## See Also

**Alloc Permanent Directory Handle 0x2222 22 18 (page 291)**, **Alloc Temporary Directory Handle 0x2222 22 19 (page 295)**, **Deallocate Directory Handle 0x2222 22 20 (page 310)**, **Set Directory Handle 0x2222 22 00 (page 491)**

# Alloc Temporary Directory Handle 0x2222 22 19

Creates a new temporary directory handle.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (19)	byte
10	SourceDirectoryHandle	byte
11	HandleName	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryHandle	byte
9	AccessRightsMask	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
157	0x9D	No Directory Handles

Decimal	Hex	Description
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

A client can have up to 255 directory handles in any combination of permanent and temporary handles. When the client signals that the task is completed, the file server automatically deallocates any temporary directory handles associated with the task.

Temporary directory handles do not need different handle names. Each separate request will create a new handle unless the 255 handle maximum has already been reached.

## See Also

**[Alloc Permanent Directory Handle 0x2222 22 18 \(page 291\)](#), [Alloc Special Temporary Directory Handle 0x2222 22 22 \(page 293\)](#), [Deallocate Directory Handle 0x2222 22 20 \(page 310\)](#), [Set Directory Handle 0x2222 22 00 \(page 491\)](#)**

# Allocate Short Directory Handle 0x2222 87 12

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (12)	byte
8	NameSpace	byte
9	DstNameSpace	byte
10	AllocateMode	word (Lo-Hi)
12	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryHandle	byte
9	VolumeNumber	byte
10	reserved[4] (0)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*EnhNWHandlePathStruct* is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

DstNameSpace is ignored unless the high bit (0x8000) is set. If the high bit is not set, the NameSpace field specifies what name space the allocation takes place in. DstNameSpace allows you to call the NCP in one name space and allocate a directory handle in another name space.

*AllocateMode* can have the following values:

Bit Number	Value	Type of Handle
0	0x00	Permanent Handle
1	0x01	Temporary Handle
2	0x02	Special Temporary Handle
3-13		reserved
14	0x4000	Activates Reply Level 2
15	0x8000	Activates DstNameSpace input parameter

If 0x4000 is set, the following reply format is used:

Offset	Content	Type
Reply header		
8	Volume	long
12	DirectoryBase	long
16	DOSDirectoryBase	long
20	NameSpace	long
24	DirectoryHandle	byte

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

## See Also

[Set Short Directory Handle 0x2222 87 09 \(page 505\)](#)

# Close File 0x2222 66

Closes a file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (66)	byte
7	Reserved	byte
8	FileHandle	byte[6] (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Unlock Error

## Remarks

*FileHandle* must be the handle returned to the client by an **Open File** or **Create File** request. After a file is closed, the file handle is no longer valid. File access requests to an invalid file handle will be disregarded by the server.

A file handle must be closed before a file can be made available to other clients. *FileHandle* requires the left most or most significant 4 bytes in Hi-Lo order.

# Commit File 0x2222 59

Ensures that all data previously written to a file has been written to the server's disk.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (59)	byte
7	Reserved	byte[1]
10	FileHandle	byte[6] (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
152	0x98	Disk Map Error
255	0xFF	Unlock Error

## Remarks

Because files must be stored on the physical storage medium before certain actions are attempted, **Commit File** provides a checkpoint that guarantees that the file has been flushed from cache and written to disk.

*FileHandle* requires the left most or most significant 4 bytes in Hi-Lo order.



# Convert Path to Dir Entry 0x2222 23 244

Converts a given path name to a directory entry.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (3+PathStringLen)	word (Hi-Lo)
9	SubFunctionCode (244)	byte
10	DirectoryHandle	byte (Lo-Hi)
11	PathStringLen	byte
12	PathString	byte[PathStringLen]

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte
9	DirectoryNumber	long (Lo-Hi)

## See Also

**Map Directory Number to Path 0x2222 23 243 (page 385)**

# Copy from One File to Another 0x2222 74

Copies data from one file on a file server to another file on the same file server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (74)	byte
7	Reserved	byte
8	SourceFileHandle	byte[6] (Hi-Lo)
14	TargetFileHandle	byte[6] (Hi-Lo)
20	SourceFileOffset	long (Hi-Lo)
24	TargetFileOffset	long (Hi-Lo)
28	BytesToCopy	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	BytesActuallyTransferred	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
148	0x94	No Write Privileges
149	0x95	File Detached
150	0x96	Server Out Of Memory

Decimal	Hex	Description
162	0xA2	IO Lock Error

## Remarks

**Copy from One File to Another** eliminates the need for a client to copy data from one file to another by reading the data from the file server, across the network to the client's workstation, and then writing the same data back to the file server across the network.

The client must have previously opened both files and must have read privileges for the source file and write privileges for the destination file. The server transfers the specified amount of information from the source to the destination file, starting at the offsets given by the client. If the end of the source file is reached before the specified number of bytes have been transferred, the transfer stops and returns Successful; however, the number of bytes actually transferred will be fewer than the number of bytes the client requested.

*SourceFileHandle* and *TargetFileHandle* require the left most or most significant four bytes in Hi-Lo order.

# Create Directory 0x2222 22 10

Creates a directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (10)	byte
10	DirectoryHandle	byte
11	DirectoryAccessMask	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
132	0x84	No Create Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

*DirectoryPath* must contain at least one element. The last element of this string will be used as the name of the newly created directory. Wildcard characters are not allowed in the new directory name. Directory names are restricted to DOS 8.3 names; longer names will be truncated.

The client must have creation privileges in the directory that will become the parent directory of the newly created directory to call **Create Directory**.

## See Also

**Delete Directory 0x2222 22 11 (page 311)**

# Create File 0x2222 67

Creates a new file in the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (67)	byte
7	DirectoryHandle	byte
8	FileAttributes	byte
9	FileNameLen	byte
10	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[6] (Hi-Lo)
14	Reserved	word (Hi-Lo)
16	FileName	byte[14]
30	FileAttributes	byte
31	FileExecuteType	byte
32	FileLen	long (Hi-Lo)
36	CreationDate	word (Hi-Lo)
38	LastAccessDate	word (Hi-Lo)
40	LastUpdateDate	word (Hi-Lo)
42	LastUpdateTime	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
128	0x80	Lock Fail
129	0x81	Out Of Handles
132	0x84	No Create Privileges
133	0x85	No Create/Delete Privileges
135	0x87	Create Filename Error
141	0x8D	Some Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

The client must have at least file creation privileges in the directory or **Create File** will be rejected. If the client has file deletion privileges in the specified directory and a file with the same name already exists in the directory, the existing file will be erased before the new file is created. If the client does not have file deletion privileges in the specified directory and a file with the same name already exists in the directory, **Create File** will fail.

The newly created file will be stamped with the date and time of its creation. *FileAttributes* will be set to the attributes specified by the client.

When it is created, the new file will be opened as an exclusive file with both read and write access requested. The actual access rights that are granted will depend on the client's file access privileges in the specified directory.

**Create File** has been replaced by **Open Create File or Subdirectory** (0x2222 87 01).

*FileHandle* requires the left most or most significant four bytes in Hi-Lo order.

## See Also

**Create New File 0x2222 77 (page 308), Erase File 0x2222 68 (page 317), Open/Create File (old) 0x2222 84 (page 395), Open/Create File or Subdirectory 0x2222 87 01 (page 398)**

# Create New File 0x2222 77

Creates a new file in the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (77)	byte
7	DirectoryHandle	byte
8	FileAttributes	byte
9	FileNameLen	byte
10	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[6] (Hi-Lo)
14	Reserved	word (Hi-Lo)
16	FileName	byte[14]
30	FileAttributes	byte
31	FileExecuteType	byte
32	FileLen	long (Hi-Lo)
36	CreationDate	word (Hi-Lo)
38	LastAccessDate	word (Hi-Lo)
40	LastUpdateDate	word (Hi-Lo)
42	LastUpdateTime	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful



Decimal	Hex	Description
128	0x80	Lock Fail
129	0x81	Out Of Handles
132	0x84	No Create Privileges
133	0x85	No Create/Delete Privileges
135	0x87	Create Filename Error
141	0x8D	Some Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

**Create New File** is the same as **Create File**, except that **Create New File** will always fail if a file with the same name already exists.

*FileHandle* requires the left most or most significant four bytes in Hi-Lo order.

## See Also

**Create File 0x2222 67 (page 306), Erase File 0x2222 68 (page 317), Open/Create File (old) 0x2222 84 (page 395), Open/Create File or Subdirectory 0x2222 87 01 (page 398)**

# Deallocate Directory Handle 0x2222 22 20

Deallocates the specified directory handle, whether the handle is temporary or permanent.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (2)	word (Hi-Lo)
9	SubFunctionCode (20)	byte
10	DirectoryHandle	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
155	0x9B	Bad Directory Handle

# Delete Directory 0x2222 22 11

Deletes the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (11)	byte
10	DirectoryHandle	byte
11	DirectoryAccessMask	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
138	0x8A	No Delete Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
159	0x9F	Directory Active
160	0xA0	Directory Not Empty
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

If the specified directory does not exist or if it contains one or more files (including subdirectories), **Delete Directory** will fail. If any other client has a directory handle pointing to the targeted directory, **Delete Directory** will fail.

The client must have access control and erase privileges to the targeted directory.

Volume directory roots cannot be deleted.

If the requesting client has directory handles pointing to the targeted directory and the directory is deleted, the associated handles are automatically deleted.

## See Also

**Create Directory 0x2222 22 10 (page 304)**

# Delete a File or Subdirectory 0x2222 87 08

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (08)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	NWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NWHandlePathStruct* is of type **NetWareHandlePathStruct** (page 609).

# Delete Trustee from Directory 0x2222 22 14

Deletes the specified trustee ID from the trustee list of the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (8 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (14)	byte
10	DirectoryHandle	byte
11	TrusteeID	long (Hi-Lo)
15	Reserved	byte
16	DirectoryPathLen	byte
17	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
252	0xFC	No Such Object
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	Failure, Hard Failure

## Remarks

The client must have access control privileges to the target directory or its parent directory.

## See Also

**Add Trustee to Directory 0x2222 22 13 (page 287), Add Extended Trustee to Directory or File 0x2222 22 39 (page 286)**

# Delete Trustee Set from File or SubDirectory 0x2222 87 11

Deletes a trustee set from a file or from a subdirectory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (11)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	ObjectIDCount	word (Lo-Hi)
12	NWHandlePathStruct	structure
xx	TrusteeStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#) and totals 307 bytes in length. TrusteeStruct starts after the 307 bytes.



# Erase File 0x2222 68

Deletes a file from the target directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (68)	byte
7	DirectoryHandle	byte
8	SearchAttributes	byte
9	FileNameLen	byte
10	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
138	0x8A	No Delete Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

The client must have file deletion privileges in the target directory or **Erase File** will fail. This request will also fail if another client is using the targeted file.

**Erase File** support wildcard characters.

## See Also

[Create File 0x2222 67 \(page 306\)](#)

# Extract a Base Handle 0x2222 22 23

Returns the server's internal form of the directory address for a workstation.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (2)	word (Hi-Lo)
10	SubFunctionCode (23)	byte
11	DirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ServerNetworkAddress	byte[10]
9	DirectoryHandle	byte[4]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
155	0x9B	Bad Directory Handle

## Remarks

*DirectoryHandle* in the request must have been previously obtained by calling **Alloc Permanent Directory Handle** (0x2222 22 18). The base Handle ID that is returned can be used when you call **Restore an Extracted Base Handle** (0x2222 22 24).

## See Also

**Restore an Extracted Base Handle 0x2222 22 24 (page 446)**

# File Migration Request 0x2222 90 150

Set or resets the file migration attributes of a file.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (150)	byte
10	Volume	long (Lo-Hi)
14	DOSDirectoryEntry	long (Lo-Hi)
16	FileMigrationState	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
251	0xFB	No Such Property

## Remarks

*FileMigrationState* can have the following values:

- 0 Ineligible for file migration
- 1 Eligible for file migration
- 2 Migrated; delete fat chains
- 3 Reset file status back to normal
- 4 Get file data back and reset file status back to normal

# File Search Continue 0x2222 63

Returns information about a file or directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (63)	byte
7	VolumeNumber	byte
8	DirectoryID	word (Hi-Lo)
10	SearchSequence	word (Hi-Lo)
12	SearchAttributes	byte
13	SearchPathLen	byte
14	SearchPath	byte[SearchPathLen]

## Reply Format (file instance)

Offset	Content	Type
Reply header		
8	SearchSequence	word (Hi-Lo)
10	DirectoryID	word (Hi-Lo)
12	Reserved	word (Hi-Lo)
14	FileName	byte[14]
28	FileAttributes	byte
29	FileMode	byte
30	FileLen	long (Hi-Lo)
34	FileCreationDate	word (Hi-Lo)
36	FileAccessDate	word (Hi-Lo)
38	FileUpdateDate	word (Hi-Lo)
40	FileUpdateTime	word (Hi-Lo)

## Reply Format (directory instance)

Offset	Content	Type
Reply header		
8	SearchSequence	word (Hi-Lo)
10	DirectoryID	word (Hi-Lo)
12	Reserved	word (Hi-Lo)
14	DirectoryName	byte[14]
28	DirectoryAttributes	byte
29	DirectoryAccessRights	byte
30	DirectoryCreationDate	word (Hi-Lo)
32	DirectoryCreationTime	word (Hi-Lo)
34	OwningObjectIdentifier	long (Hi-Lo)
38	Reserved	word (Hi-Lo)
40	DirectoryStamp (0xD1D1)	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	No Files Found

## Remarks

**File Search Continue** is called iteratively after calling **File Search Initialize** (0x2222 62).

The information that is returned is a direct copy of a Novell server's internal file header. **File Search Continue** does not return the ID of the file owner.

*SearchSequence* is the file offset in the directory file. The file server increments this number by one in the reply. Setting this field to -1 will restart the search. When *SearchSequence* is set to -1, *SearchAttributes* can be modified to alter the search mask.

The DOS DIR command is implemented in NetWare through **File Search Continue**. The non-directory files are obtained first by calling **File Search Init**. Then, normal, non-directory files are returned by repeatedly calling **File Search Continue**.

When all of the non-directory files have been returned, the server returns No Files Found and sets *SearchSequence* to -1. The client then calls **File Search Continue** with *SequenceNumber* set to -1 and the *SearchAttributes* set to return directory files. **File Search Continue** is then called repeatedly until all of the subdirectories are returned.

## See Also

[Scan Directory Information 0x2222 22 02 \(page 460\)](#), [File Search Initialize 0x2222 62 \(page 324\)](#)

# File Search Initialize 0x2222 62

Initiates a search for files in a directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (62)	byte
7	DirectoryHandle	byte
8	PathLen	byte
9	Path	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte
9	DirectoryID	word (Hi-Lo)
11	SearchSequenceNumber (-1)	word (Hi-Lo)
13	DirectoryAccessRights	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	No Files Found



## Remarks

**File Search Initialize** returns the *DirectoryID* and *DirectoryAccessRights* of the directory specified by *DirectoryHandle* and *Path*. The *VolumeNumber* and *DirectoryID* can then be used in **File Search Continue** (0x2222 63) to locate files or subdirectories.

Beginning with NetWare 2.0a, the workstation shell uses this request, in conjunction with **File Search Continue**, to perform file and directory searches. Both requests were added to accommodate the DOS OS. Older requests relied on the server to keep a search context associated with a directory handle. When End Of Task was received from the workstation, the directory handle was deleted, which caused the search context to be lost. Since DOS sends "pseudo" End Of Task messages in a command file loop, directory handles were deleted, which caused the older requests to fail.

With **File Search Initialize**, the search context is the file identifier in the Novell server file system; the context is no longer based on directory handles. Consequently, Novell's server implementation does not maintain a context block for searches performed by calling **File Search Initialize**.

**File Search Initialize** is replaced by the NetWare v3.11 **Initialize Search** (0x2222 87 02) request.

## See Also

**File Search Continue 0x2222 63 (page 321), Initialize Search 0x2222 87 02 (page 384)**

# Generate Directory Base and Volume Number 0x2222 87 22

Generates a directory base and volume number.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (22)	byte
8	srcNameSpace	byte
9	dstNameSpace	byte
10	dstNSIndicator	word
12	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NSDirectoryBase	long (Lo-Hi)
12	DOSDirectoryBase	long (Lo-Hi)
16	VolumeNumber	byte

## Parameters

*srcNameSpace*

(Request) Specifies the source name space used to parse path information.

*dstNameSpace*

(Request) Specifies the name space for the returned directory base and volume number information if *dstNSIndicator* is set to 'Jn'; otherwise, this parameter is ignored.

*dstNSIndicator*

(Request) Specifies to return the directory base and volume number in the name space specified by *dstNameSpace* if this parameter is set to 'Jn'; otherwise, the directory base and volume number information will be returned in the name space specified by *srcNameSpace*.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NWHandlePathStruct* is of type [NetWareHandlePathStruct](#) (page 609).

If *dstNSIndicator* is set to 'In', **Generate Directory Base and Volume Number** will return a directory base for a name space that you are not currently in.

# Get Current Size of File 0x2222 87 66

Determines the length of a file that the client has opened.

**NetWare Servers:** 6.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (66)	byte
8	FileHandle	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentFileSize	UINT64 (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
253	0xFD	Bad Station Number

## Remarks

Note that *CurrentFileSize* in the reply can be 64 bits. Requests made to NSS volumes support returning true 64-bit offsets. Requests made to traditional volumes support returning 32-bit offsets.

64-bit file sizes can be obtained from NSS volumes by using the following NCPs:

- ◆ [Open/Create File or Subdirectory 0x2222 87 01 \(page 398\)](#)
- ◆ [Open/Create File or Subdirectory 0x2222 87 30 \(page 401\)](#)
- ◆ [Open/Create File or Subdirectory with Callback 0x2222 87 32 \(page 404\)](#)
- ◆ [Open/Create File or Subdirectory II with Callback 0x2222 87 33 \(page 407\)](#)
- ◆ [Search for File or Subdirectory 0x2222 87 03 \(page 480\)](#)

- ♦ [Obtain File or Subdirectory Information 0x2222 87 06 \(page 392\)](#)
- ♦ [Search for File or Subdirectory Set 0x2222 87 20 \(page 482\)](#)
- ♦ [Scan Salvageable Files 0x2222 87 16 \(page 472\)](#)
- ♦ [Get Effective Directory Rights 0x2222 87 29 \(page 343\)](#)

Pass in a *ReturnInfoMask* that includes at least the following bits (see “[Extended ReturnInfoMask Values](#)” on [page 638](#) for a full description):

0x80000000 RNewStyle  
0x04000000 R64BitFileSize

**Get Current Size of File** can be called by clients who are cooperatively sharing a file that needs to be extended.

When a shared file needs to be extended, the client that is extending the file needs to lock the area of the file that will be affected for their own exclusive use. This can be done by locking the entire file or by locking the section of the file that is beyond the current known EOF.

After locking the file, the client that is extending the file must call **Get Current Size of File** to determine the current file length. If the file has already been extended by some other client, **Get Current Size of File** reveals the length of the extension to the current client so that the file can be extended further, if needed. The file can then be unlocked.

Only by using this method of extending a shared file can a client properly lock the current end of a file for exclusive use; otherwise, another client could extend the file at any time.

*FileHandle* requires the input to be in Hi-Lo order.

## See Also

[Clear Physical Record 0x2222 87 69 \(page 1168\)](#), [Lock Logical Record Set 0x2222 108 \(page 1180\)](#), [Lock Physical Record Set 0x2222 110 \(page 1182\)](#), [Log Physical Record 0x2222 87 67 \(page 1189\)](#), [Release Physical Record 0x2222 87 68 \(page 1204\)](#)

# Get Current Size of File 0x2222 71

Determines the length of a file that the client has opened.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (71)	byte
7	Reserved	byte
8	FileHandle	byte[6] (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentFileSize	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle

## Remarks

**Get Current Size of File** can be called by clients who are cooperatively sharing a file that needs to be extended.

When a shared file needs to be extended, the client that is extending the file needs to lock the area of the file that will be affected for their own exclusive use, which can be done by locking the entire file or by locking the section of the file that is beyond the current known EOF.

After locking the file, the client that is extending the file must call **Get Current Size of File** to determine the current file length. If the file has already been extended by some other client, **Get Current Size of File** will reveal the length of the extension to the current client so that the file can be extended further, if needed. The file can then be unlocked.

*FileHandle* requires the left most or most significant four bytes in Hi-Lo order.

Only by using this method of extending a shared file can a client properly lock the current end of a file for exclusive use; otherwise, another client could extend the file at any time.

# Get Directory Disk Space Restriction 0x2222 22 35

Scans for the amount of disk space assigned to all directories between the current directory and the root directory and returns information about the restrictions along the directory path.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (35)	byte
10	DirHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfEntries	byte (for each entry)
9	Level	byte
10	Max	long (Lo-Hi)
14	Current	long (Lo-Hi)

## Parameters

*Level*

(Reply) Specifies the distance from the directory to the root.

*Max*

(Reply) Specifies the maximum amount of space assigned to a directory.

*Current*

(Reply) Specifies the amount of space assigned to the directory minus the amount of space used by the directory and its subdirectories.



# Return Values

Decimal	Hex	Description
0	0x00	Successful

# Remarks

To find the actual amount of space available for a directory, scan all the current entries and use the smallest entry. Directories that have no restrictions will not return any information. If no entries are returned, no space restrictions exist for the specified directory. All restrictions are in 4 KB blocks.

When *Max* is 0x7FFFFFFF, there is no restriction on an entry; however, you can still calculate the space in use by subtracting *Current* from *Max*.

When *Max* is negative, the limit is zero. When *Current* is negative, the amount of space assigned to the directory is really zero. These two fields are allowed to be negative so you can still generate a valid "IN USE" value.

# See Also

**Add User Disk Space Restriction 0x2222 22 33 (page 290), Remove User Disk Space Restrictions 0x2222 22 34 (page 438), Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366), Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476), Set Directory Disk Space Restriction 0x2222 22 36 (page 487), Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Get Directory Disk Space Restriction 0x2222 87 39

Scans for the amount of disk space assigned to all directories between the current directory and the root directory and returns information about the restrictions along the directory path.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (39)	byte
8	NameSpace	byte
9	reserved	byte[2]
11	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfEntries	byte (for each entry)
9	DirDiskSpaceResList	structure

## Parameters

*NumberOfEntries*

(Reply) Specifies the number of *DirDiskSpaceResList* structures that follow.

*Max*

(Reply) Specifies the maximum amount of space assigned to a directory.

*Current*

(Reply) Specifies the amount of space assigned to the directory minus the amount of space used by the directory and its subdirectories.

# Return Values

Decimal	Hex	Description
0	0x00	Successful

# Remarks

To find the actual amount of space available for a directory, scan all the current entries and use the smallest entry. Directories that have no restrictions will not return any information. If no entries are returned, no space restrictions exist for the specified directory. All restrictions are in 4 KB blocks.

When *Max* is 0x7FFFFFFF, there is no restriction on an entry; however, you can still calculate the space in use by subtracting *Current* from *Max*.

When *Max* is negative, the limit is zero. When *Current* is negative, the amount of space assigned to the directory is really zero. These two fields are allowed to be negative so you can still generate a valid "IN USE" value.

NWHandlePathStruct is of type [NetWareHandlePathStruct \(page 609\)](#).

The target must be a subdirectory or an error will be returned.

There will be one *DirDiskSpaceRestList* for each entry back to the root from the current entry.

# See Also

[Add User Disk Space Restriction 0x2222 22 33 \(page 290\)](#), [Remove User Disk Space Restrictions 0x2222 22 34 \(page 438\)](#), [Get Object Disk Usage and Restrictions 0x2222 22 41 \(page 366\)](#), [Scan Volume's User Disk Restrictions 0x2222 22 32 \(page 476\)](#), [Set Directory Disk Space Restriction 0x2222 22 36 \(page 487\)](#), [Get Directory Disk Space Restriction 0x2222 22 35 \(page 332\)](#), [Scan Directory Disk Space 0x2222 22 40 \(page 455\)](#)

# Get Directory Entry 0x2222 22 31

Returns the directory entry that is pointed to by the specified directory handle.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (31)	byte
10	DirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Subdirectory	long (Lo-Hi)
12	Attributes	long (Lo-Hi)
13	UniqueID	byte
14	Flags	byte
15	NameSpace	byte
16	DirectoryNameLen	byte
17	DirectoryName	byte[12]
29	CreationDateAndTime	long (Lo-Hi)
33	OwnerID	long (Hi-Lo)
37	LastArchivedDateAndTime	long (Lo-Hi)
41	LastArchivedID	long (Hi-Lo)
45	LastModifiedDateAndTime	long (Lo-Hi)
49	NextTrusteeEntry	long (Hi-Lo)
53	Reserved	byte[48]
101	MaximumSpace	word (Lo-Hi)

Offset	Content	Type
103	InheritedRightsMask	word (Lo-Hi)
105	Undefined	byte[28]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
137	0x89	No Search Privileges
191	0xBF	Invalid Name Space
251	0xFB	386 File Structure Not Supported On Server

## Remarks

**Get Directory Entry** is useful for getting information about the volume root entry.

## See Also

**Set Directory Handle 0x2222 22 00 (page 491), Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Get Directory Information 0x2222 22 45

Returns the real size information for a 386 directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (45)	byte
10	DirHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	TotalBlocks	long (Lo-Hi)
12	AvailableBlocks	long (Lo-Hi)
16	TotalDirEntries	long (Lo-Hi)
20	AvailableDirEntries	long (Lo-Hi)
24	Reserved	byte[4]
28	SectorsPerBlock	byte
29	VolumeNameLen	byte
30	Volume	byte[VolumeNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
155	0x9B	Bad Directory Handle

## Remarks

The old NCP requests cannot handle volumes bigger than 256 MB.

**Get Directory Information** also includes space limitations on the user and volume when calculating the space available.

## See Also

**Set Directory Information 0x2222 22 25 (page 493), Get Volume and Purge Information 0x2222 22 44 (page 375)**

# Get Directory Path 0x2222 22 01

Returns the full directory path of the directory pointed to by one of its directory handles.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (1)	byte
10	TargetDirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryPathLen	byte
9	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error

## Remarks

The directory path string that is returned will contain a path name in the following format:

`volume name:directory/subdirectory/...`

The returned string will not contain the files server's name and will be no longer than 255 bytes.



## See Also

[Set Directory Handle 0x2222 22 00 \(page 491\)](#)

# Get Effective Directory Rights 0x2222 22 03

Returns the access rights the client has for the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (3 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (3)	byte
10	DirectoryHandle	byte
11	DirectoryPathLen	byte
12	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Hard Failure, Failure

## Remarks

The returned *EffectiveRightsMask* indicates which of the eight possible directory rights the client has in the targeted directory. An *EffectiveRightsMask* of zero indicates that the client has no rights in the target directory.

## See Also

**Get Effective Rights for Directory Entry 0x2222 22 42 (page 344)**

# Get Effective Directory Rights 0x2222 87 29

Returns the access rights the client has for the specified directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (29)	byte
8	NameSpace	byte
9	DestNameSpace	byte
10	SearchAttributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	MyEffectiveRights	word (Lo-Hi)
10	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NWHandlePathStruct* is of type [NetWareHandlePathStruct](#) (page 609).

# Get Effective Rights for Directory Entry 0x2222 22 42

Returns the calling object's effective rights to the specified entry.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (3 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (42)	byte
10	DirHandle	byte
11	PathLen	byte
12	Path	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	AccessRights	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Disk Map Error
156	0x9C	Invalid Path

## Remarks

**Get Effective Rights for Directory Entry** works for directories and files.

## See Also

**Get Effective Directory Rights 0x2222 22 03 (page 342)**, **Get Object Effective Rights for Directory Entry 0x2222 22 50 (page 368)**

# Get Extended Volume Information 0x2222 22 51

Returns the full volume information about a mounted volume.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (2)	word (Hi-Lo)
9	SubFuncCode (51)	byte
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	VollInfoReplyLen	word (Lo-Hi)
10	VolumeInfo ( <a href="#">VollInfoStructure (page 627)</a> )	structure
10 + VollInfoReplyLen	VolNameLen	byte
11 + VollInfoReplyLen	VolumeName	byte[VolNameLen]

## Parameters

*volumeNumber*

(Request) Specifies the volume number to start with.

*VollInfoReplyLen*

(Reply) Specifies the size of VollInfoStructure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

Decimal	Hex	Description
152	0x98	Disk Map Error
255	0xFF	Failure

## Remarks

**Get Effective Rights for Directory Entry** works for directories and files.

## See Also

**Get Effective Directory Rights 0x2222 22 03 (page 342)**, **Get Object Effective Rights for Directory Entry 0x2222 22 50 (page 368)**

# Get File Information 0x2222 87 31

Returns various sets of information about a file from a NetWare file handle or about a directory from a 1-byte directory handle.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (31)	byte
8	FileOrDirHandle	byte[6] (Hi-Lo)
14	HandleInfoLevel	byte
15	NameSpace	byte

## Reply Format (HandleInfoLevel = 0)

Get Limited Information from a File Handle:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DataStream	long (Lo-Hi)

## Reply Format (HandleInfoLevel = 1)

Get Limited Information from a File Handle Using a Name Space:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DataStream	long (Lo-Hi)

## Reply Format (HandleInfoLevel = 2)

Get Information from a File Handle:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	NameSpace	long (Lo-Hi)
24	<b>DataStream</b>	long (Lo-Hi)

## Reply Format (HandleInfoLevel = 3)

Get Information from a Directory Handle:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	NameSpace	long (Lo-Hi)

## Reply Format (HandleInfoLevel = 4)

Get Complete Information from a Directory Handle:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	NameSpace	long (Lo-Hi)
24	ParentDirectoryBase	long (Lo-Hi)
28	ParentDOSDirectoryBase	long (Lo-Hi)



# Reply Format (HandleInfoLevel = 5)

Get Complete Information from a File Handle:

Offset	Content	Type
Reply header		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	NameSpace	long (Lo-Hi)
24	DataStream	long (Lo-Hi)
28	ParentDirectoryBase	long (Lo-Hi)
32	ParentDOSDirectoryBase	long (Lo-Hi)

## Parameters

*FileOrDirHandle*

(Request) Specifies either a file handle or directory handle for which to return information.

*HandleInfoLevel*

(Request) Specifies which type of information to return.

*NameSpace*

(Request) Specifies the name space for which to return information; only used when *HandleInfoLevel* is set to 1.

*Volume*

(Reply) Specifies the volume of the file handle.

*DirectoryBase*

(Reply) Specifies the directory entry number.

*DOSDirectoryBase*

(Reply) Specifies the DOS directory entry number.

*NameSpace*

(Reply) Specifies the name space associated with the *DirectoryBase* and is based on the file or directory handle passed in the request.

*DataStream*

(Reply) Specifies the data stream number if the name space is Macintosh (see “**DataStream Values**” on page 633).

*ParentDirectoryBase*

(Reply) Specifies the parent directory entry number of the file or directory.

*ParentDOSDirectoryBase*

(Reply) Specifies the DOS parent directory entry number of the file or directory.

## Remarks

*FileOrDirHandle* requires the left most or most significant four bytes in Hi-Lo order.

# Get Full Path String 0x2222 87 28

NetWare Servers: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (28)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	PathCookie	structure
20	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	PathCookie	structure
18	PathComponentSize	word (Lo-Hi)
20	PathComponentCount	word (Lo-Hi)
22	PathComponent[]	structure

## Parameters

*PathCookie*

(Reply) Specifies the sequence path components that are too long to fit in a reply packet.

*PathComponentSize*

(Reply) Specifies the total byte size of *PathComponent*.

*PathComponentCount*

(Reply) Specifies the number of components in the reply packet.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The path returned by **Get Full Path String** is returned in reverse order, with the root being the last component and the current directory being the first component.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

# Get Huge NS Information 0x2222 87 26

Returns the huge name space information.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (26)	byte
8	NameSpace	byte
9	VolumeNumber	byte
10	DirectoryBase	long (Lo-Hi)
14	HugeMask	long (Lo-Hi)
18	HugeStateInfo	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextHugeStateInfo	structure
24	HugeDataLen	long (Lo-Hi)
28	HugeData[]	byte

## Parameters

*HugeStateInfo*

Is used only by the NFS name space.

*HugeDataLen*

Specifies the length of the data that will be returned in the reply buffer.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Get Huge NS Information** is called only when the name space has indicated that there is huge information available (by calling **Query NS Information Format** 0x2222 87 23).

*HugeStateInfo* is a 16-byte structure that contains information used to help the name space in transferring huge name space information across the wire. For the initial request, all 16 bytes should be set to zero. All subsequent requests will set *HugeStateInfo* using information from the *NextStateInfo* field.

## See Also

**Set Huge NS Information 0x2222 87 27 (page 502)**

# Get Mount Volume List 0x2222 22 52

Returns a list of mounted volumes.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (52)	byte
10	StartVolumeNumber	long (Lo-Hi)
14	VolumeRequestFlags	long (Lo-Hi)
18	NameSpace	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ItemsInPacket	long
12	NextVolumeNumber	long
16+	VolMntStruc	structure[ItemsInPacket]

## Parameters

*StartVolumeNumber*

(Request) Specifies the starting volume number (initial value is zero).

*NameSpace*

(Request) Specifies the name space number for which you are requesting mounted volumes.

*NextVolumeNumber*

(Request) Specifies the next starting volume number.

*VolumeRequestFlags*

(Request) Specifies whether to return the volume name with the volume number:

0 Return the volume name with the volume number.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Get Mount Volume List** returns mounted volumes for only the desired name space that is loaded.

If *NextVolumeNumber* is zero, the list is complete.

If *VolumeRequestFlags* is set to zero, the **VolMntStructWithName** (page 632) will be returned. Otherwise, the **VolMntStruct** (page 631) will be returned.



# Get NS Information 0x2222 87 19

Returns information for the specified name space.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (19)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	reserved (0)	byte
11	VolumeNumber	byte
12	DirectoryBase	long (Lo-Hi)
16	NSInfoBitMask	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NSSpecificInfo[512]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Get NS Information** is passed to the name space NLM and is an expensive time user on the server.

*NSSpecificInfo* is defined as:

```
BYTE   FileName&LengthByte[13]:
LONG   FileAttributes; (Lo-Hi)
WORD   CreateDate; (Lo-Hi)
WORD   CreateTime; (Lo-Hi)
```

LONG	OwnerID; (Hi-Lo)
WORD	ArchiveDate; (Lo-Hi)
WORD	ArchiveTime; (Lo-Hi)
LONG	ArchiveID; (Hi-Lo)
WORD	ModifyDate; (Lo-Hi)
WORD	ModifyTime; (Lo-Hi)
LONG	ModifyID; (Hi-Lo)
WORD	LastAccessDate; (Lo-Hi)
LONG	InheritedRightsMask; (Lo-Hi)
LONG	MaximumSpace; (Lo-Hi)

## See Also

**Set NS Information 0x2222 87 25 (page 504)**

# Get Name Space Directory Entry 0x2222 22 48

Returns a directory entry associated with any name space supported on the volume.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (48)	byte
10	VolumeNumber	byte
11	DOS Sequence	long
15	NameSpace	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SequenceNumber	long (DOS File Entry)
12	Subdirectory	long
16	Attributes	long
20	UniqueID	byte
21	Flags	byte
22	NameSpace	byte
23	NameLen	byte
24	Name	byte[12]
36	CreationDateAndTime	long
40	OwnerID	long (Hi-Lo)
44	LastArchivedDateAndTime	long
48	LastArchivedID	long (Hi-Lo)
52	LastUpdatedDateAndTime	long

Offset	Content	Type
56	LastUpdatedID	long (Hi-Lo)
60	FileSize	long
64	Reserved	byte[44]
108	InheritedRightsMask	word
110	LastAccessedDate	word
112	Reserved	byte[28]

## Reply Format (DOS Subdirectory Entry)

Offset	Content	Type
Reply header		
16	Attributes	long
20	UniqueID	byte
21	Flags	byte
22	NameSpace	byte
23	DirectoryNameLen	byte
24	DirectoryName	byte[12]
36	CreateDateAndTime	long
40	OwnerID	long (Hi-Lo)
44	LastArchivedDateAndTime	long
48	LastArchivedID	long (Hi-Lo)
52	LastModifiedDateAndTime	long
56	NextTrusteeEntry	long
60	Reserved	byte[48]
108	MaximumSpace	long
112	InheritedRightsMask	word
112	Undefined	byte[26]

## Reply Format (Macintosh Name Space Entry)

Offset	Content	Type
Reply header		
16	MACFileAttributes	long
20	MACUniqueID	byte
21	MACFlags	byte
22	MACMyNameSpace	byte
23	MACFileNameLen	byte
24	MACFileName	byte[32]
56	ResourceFork	long
60	ResourceForkSize	long
64	FinderInfo	byte[32]
96	ProDosInfo	byte[6]
102	Reserved	byte[38]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
137	0x89	No Search Privileges
152	0x98	Invalid Volume
191	0xBF	Invalid Name Space

## Remarks

Usually, you would get the entry's sequence number by scanning for it in the DOS name space.

The name spaces that are currently defined include:

- 0 DOS
- 1 Macintosh

DOS Sequence in the request buffer at offset 11 and SequenceNumber in the reply buffer at offset 8 are referred to as either a DirectoryBase or a DirectoryHandle in NetWare 386, which is important if you are implementing an API that uses the terms DirectoryBase or DirectoryHandle rather than DOS Sequence or SequenceNumber.

## See Also

[Get Name Space Information 0x2222 22 47 \(page 363\)](#)

# Get Name Space Information 0x2222 22 47

Returns name space and data stream information for the file server and its volumes.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (47)	byte
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	DefinedNameSpaces	byte (for each Name Space)
9	NSNameLen	byte
10	NameSpaceName	byte[NSNameLen]
10 + DefinedNameSpaces + NSNameLen	DefinedDataStreams	byte (for each Data Stream)
11 + DefinedNameSpaces + NSNameLen	AssociatedNameSpace	byte
12 + DefinedNameSpaces + NSNameLen	DSNameLen	byte
13 + DefinedNameSpaces + NSNameLen	DataStreamName	byte[DSNameLen]
13 + DefinedNameSpaces + 2*DefinedDataStreams + NSNameLens + DSNameLens	LoadedNameSpaces	byte
14 + DefinedNameSpaces + 2*DefinedDataStreams + NSNameLens + DSNameLens	IndexNumber	byte[LoadedNameSpaces]
14 + DefinedNameSpaces + 2*DefinedDataStreams + NSNameLens + DSNameLens + LoadedNameSpaces	VolumeNameSpaces	byte

Offset	Content	Type
15 + DefinedNameSpaces + 2*DefinedDataStrems + NSNameLens + DSNameLens + LoadedNameSpaces	IndexNumber	byte[VolumeNameSpaces]
15 + DefinedNameSpaces + 2*DefinedDataStrems + NSNameLens + DSNameLens + LoadedNameSpaces + VolumeNameSpaces	VolumesDataStreams	byte
15 + DefinedNameSpaces + 2*DefinedDataStrems + NSNameLens + DSNameLens + LoadedNameSpaces + VolumeNameSpaces	IndexNumber	byte[VolumeDataStreams]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*IndexNumber* associated with *LoadedNameSpaces* is a list of the possible name spaces that are actually loaded.

*IndexNumber* associated with *VolumeNameSpaces* is a list of the name spaces that are being used on the specified volume.

*IndexNumber* associated with *VolumeDataStreams* is a list of the data streams that are being used on the specified volume.

## See Also

**Get Name Space Directory Entry 0x2222 22 48 (page 359)**



# Get Name Spaces Loaded List from Volume Number 0x2222 87 24

Returns the name spaces that are loaded on the specified volume.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (24)	byte
8	reserved (0)	word (Lo-Hi)
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfNSLoaded	word (Lo-Hi)
10	NSLoadList[]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NSLoadList* is a byte array with *NumberOfNSLoaded* elements, with each element being the name space value (DOS, MACINTOSH, etc.) that is loaded.

# Get Object Disk Usage and Restrictions 0x2222 22 41

Scans a user's disk restrictions for a volume and returns the amount of space currently being used.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (41)	byte
10	VolumeNumber	byte
11	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	Restriction	long (Lo-Hi)
12	InUse	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Invalid Volume

## Remarks

The space values are returned in 4 KB blocks.

If the restriction is 0x40000000, there is no restriction for the object.

**Get Object Disk Usage and Restrictions** succeeds even if the object ID is invalid; it will return no restrictions and no space being used.

## See Also

**Add User Disk Space Restriction 0x2222 22 33 (page 290), Remove User Disk Space Restrictions 0x2222 22 34 (page 438), Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476), Set Directory Disk Space Restriction 0x2222 22 36 (page 487), Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Get Object Effective Rights for Directory Entry 0x2222 22 50

Returns an object's effective access rights to a specified directory or file.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (7 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (50)	byte
10	ObjectID	long (Hi-Lo)
14	DirHandle	byte
15	PathLen	byte
16	Path	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	AccessRights	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
155	0x9B	Invalid Directory Error
156	0x9C	Invalid Path

## Remarks

The client must be either a supervisor, a manager of the object for which rights are requested, or the object itself.

**Get Object Effective Rights for Directory Entry** is similar to **Get Effective Rights for Directory Entry 0x2222 22 42**); however, **Get Object Effective Rights for Directory Entry** allows you to specify an object for which rights are returned.

## See Also

**[Get Effective Rights for Directory Entry 0x2222 22 42 \(page 344\)](#)**

# Get Path Name of a Volume-Directory Number Pair 0x2222 22 26

Returns the directory path for a volume number/directory entry number pair.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (4)	word (Hi-Lo)
9	SubFunctionCode (26)	byte
10	VolumeNumber	byte
11	DirectoryEntryNumber	word

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryPathLen	byte
9	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error

# Get Path String from Short Directory Handle 0x2222 87 21

Returns a path string from a short directory handle.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (21)	byte
8	NameSpace	byte
9	ShortDirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	PathLen	byte
9	Path[]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Get Reference Count from Dir Entry Number 0x2222 90 10

Returns a reference count for the specified directory entry number.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFunctionCode (10)	byte
10	Volume	long (Lo-Hi)
14	DirBase	long (Lo-Hi)
18	NameSpace	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ReferenceCount	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
152	0x98	Disk Map Error (Invalid Volume)
156	0x9C	Invalid Path



# Get Reference Count from Dir Handle 0x2222 90 11

Returns a reference count for the specified directory handle.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (11)	byte
10	DirectoryHandle	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ReferenceCount	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
136	0x88	Invalid Directory Handle

# Get Sparse File Data Block Bit Map 0x2222 85

Returns a bit map that shows which blocks contain data and which are empty.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (85)	byte
7	SubFuncStrucLen (10)	word (Hi-Lo)
9	Handle	byte[6]
15	StartingOffset	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	AllocationBlockSize	long (Lo-Hi)
12	Reserved	byte[4]
16	BitMap (1 bit for each block)	byte[512]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle

## Remarks

There is one bit for each block in the parse file. One means that there is data in that block; zero means that there is no data in that block.

# Get Volume and Purge Information 0x2222 22 44

Returns the real volume information for a 386 volume and returns information about deleted files.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (44)	byte
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	TotalBlocks	long (Hi-Lo)
12	FreeBlocks	long (Lo-Hi)
16	PurgeableBlocks	long (Lo-Hi)
20	NotYetPurgeableBlocks	long (Lo-Hi)
24	TotalDirEntries	long (Lo-Hi)
28	AvailableDirEntries	long (Lo-Hi)
32	Reserved	byte[4]
36	SectorsPerBlock	byte
37	VolumeNameLen	byte
38	VolumeName	byte[VolumeNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Old NCP requests cannot handle a volume that is bigger than 256 MB.

If the volume number specified in the request buffer is not mounted, a Successful is returned; but all data fields in the reply buffer are set to zero.

# Get Volume Info with Handle 0x2222 22 21

Returns information about the physical limitations of one of the server's volumes.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (21)	byte
10	DirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SectorsPerCluster	word (Hi-Lo)
10	TotalVolumeClusters	word (Hi-Lo)
12	AvailableClusters	word (Hi-Lo)
14	TotalDirectorySlots	word (Hi-Lo)
16	AvailableDirectorySlots	word (Hi-Lo)
18	VolumeName	byte[16]
34	RemovableFlag	word (Hi-Lo)

## Parameters

*SectorsPerCluster*

(Reply) Specifies how many 512-byte sectors are contained in each cluster.

*TotalVolumeClusters*

(Reply) Specifies how many clusters make up the server volume.

*AvailableClusters*

(Reply) Specifies how many clusters are not currently in use.

### *TotalDirectorySlots*

(Reply) Specifies how many physical directory entries the volume contains (optional).

### *AvailableDirectorySlots*

(Reply) Specifies how many of the total directory entries on the volume are still available for use (optional).

### *VolumeName*

(Reply) Specifies the name of the volume whose statistics are being reported (NULL-padded).

### *RemovableFlag*

(Reply) Specifies if the volume is on a fixed or removable media:

0            Volume is on fixed media

0xFFFF    Volume is on removable (mountable) media

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

Volumes use logical sector sizes of 512 bytes per sector. If the physical media uses a different sector size, the server must perform the appropriate mappings. Volume space is allocated in groups of sectors called clusters.

If *TotalDirectorySlots* and *AvailableDirectorySlots* are meaningless under a given server's implementation, set these parameters to 0xFFFF.

## See Also

**Get Volume Info with Number 0x2222 18 (page 379)**

# Get Volume Info with Number 0x2222 18

Returns information about the physical limitations of one of the server's volumes.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (18)	byte
7	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SectorsPerCluster	word (Hi-Lo)
10	TotalVolumeClusters	word (Hi-Lo)
12	AvailableClusters	word (Hi-Lo)
14	TotalDirectorySlots	word (Hi-Lo)
16	AvailableDirectorySlots	word (Hi-Lo)
18	VolumeName	byte[16]
34	RemovableFlag	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Disk Map Error

## Remarks

**Get Volume Info with Number** allows a client to check the physical space available on a volume without having to determine which mounted volume number the client's directory handle points to.

## See Also

[Get Volume Info with Handle 0x2222 22 21 \(page 377\)](#)



# Get Volume Name 0x2222 22 06

Returns a volume's name from the given volume number.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (6)	byte
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNameLen	byte
9	VolumeName	byte[VolumeNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
255	0xFF	Failure

## Remarks

**Get Volume Name** can be called to scan volumes and determine the volume (mount) numbers and volume names of all volumes that are currently mounted on the file server.

If you want to scan the file server for such information, start with volume number zero and scan upward until Failure is returned.

If a volume that corresponds to the volume number has not been mounted, **Get Volume Name** will still return Successful, but the returned *VolumeNameLen* will be zero, which indicates that no corresponding volume is currently mounted but the volume mount slot is potentially valid.

## See Also

**Get Volume Number 0x2222 22 05 (page 383)**

# Get Volume Number 0x2222 22 05

Returns a volume's number from the given volume name.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (2 + VolumeNameLen)	word (Hi-Lo)
9	SubFunctionCode (5)	byte
10	VolumeNameLen	byte
11	VolumeName	byte[VolumeNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error

## Remarks

Volume numbers are required by several requests, including the requests to return a volume's usage statistics and to return an object's trustee directory paths.

## See Also

**Get Volume Name 0x2222 22 06 (page 381)**

# Initialize Search 0x2222 87 02

Initializes the search for a file or subdirectory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (02)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte
9	DirectoryNumber	long (Lo-Hi)
13	EntryNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

This search request is a stateless search.

*SearchSequenceDefinition* is a 9-byte field, which contains a 1-byte volume number, a 4-byte directory number, and a 4-byte current directory number. *SearchSequence* is generated only by the server on each completed search request.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

# Map Directory Number to Path 0x2222 23 243

Maps a given directory number to a path.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStructLen (7)	word (Hi-Lo)
9	SubFunctionCode (243)	byte
10	VolumeNumber	byte
11	DirectoryNumber	long (Lo-Hi)
15	NameSpace	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Path	byte[]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The path is returned as a group of components. Each directory, subdirectory, or file in the path is considered to be a component. Each component is length preceeded and is followed by the next component. For example, *pathName* returns the users/jdoe/working directory as:

```
5users4jdoe6working
```

The volume name is not returned as a component in the path.

## See Also

[Convert Path to Dir Entry 0x2222 23 244 \(page 301\)](#)

# Modify DOS Attributes on a File or Subdirectory 0x2222 87 35

Sets the DOS attributes field on an entry with the use of an apply mask.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (35)	byte
8	NameSpace	byte
9	Flags	byte
10	SearchAttributes	word (Lo-Hi)
12	AttributeMask	long (Lo-Hi)
16	Attributes	long (Lo-Hi)
20	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	ItemsChecked	long (Lo-Hi)
12	ItemsChanged	long (Lo-Hi)
16	AttributeValidFlag	long (Lo-Hi)
20	NewAttributes	long (Lo-Hi)

## Parameters

*ItemsChecked*

(Reply) Specifies the number of items that were scanned according to the input criteria.

*ItemsChanged*

(Reply) Specifies the number of items whose attributes were changed.

*AttributeValidFlag*

(Reply) Specifies whether *NewAttributes* is valid.

*NewAttributes*

(Reply) Specifies the new value of the entry's attributes.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

A client doesn't have to get the old attributes before calling **Modify DOS Attributes on a File or Subdirectory**.

**Modify DOS Attributes on a File or Subdirectory** is similar to **Modify File or Subdirectory DOS Information 0x2222 87 07** but has the additional capability to modify files with a wildcard.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

If AllowWildCardsBit is set in *Flags*, wildcards can be used in *NWHandlePathStruct*. If wildcards are permitted, only the first matching entry will be returned in the reply packet.

Even if *AttributeValidFlag* is not valid, the entry's attributes were still changed.

## See Also

[Modify File or Subdirectory DOS Information 0x2222 87 07 \(page 389\)](#)



# Modify File or Subdirectory DOS Information 0x2222 87 07

Modifies DOS information while the client is in another name space.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (07)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	SearchAttributes	word (Lo-Hi)
12	ModifyDOSInfoMask	long (Lo-Hi)
16	ModifyDOSInfoStruct	structure
54	NWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Modify File or Subdirectory** will not change the name of a file or subdirectory. Passing 0x1 to *ModifyDOSInfoMask* has no effect.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct](#) (page 609).

## See Also

[Modify DOS Attributes on a File or Subdirectory 0x2222 87 35](#) (page 387)

# Modify Maximum Rights Mask 0x2222 22 04

Modifies the maximum rights mask associated with a directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (5 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (4)	byte
10	DirectoryHandle	byte
11	RightsGrantMask	byte
12	RightsRevokeMask	byte
13	DirectoryPathLen	byte
14	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

The directory's maximum access rights mask is first ANDed with the NOT of the specified *RightsRevokeMask*. This operation removes any rights included in the *RightsRevokeMask* from the directory's maximum access rights mask. The result of this operation is then ORed with the

specified *RightsGrantMask*. This operation adds the rights included in the *RightsGrantMask* to the directory's maximum access rights mask. The result is the directory's new maximum access rights mask.

Any client that has access control rights to either the target directory or its parent directory can successfully call **Modify Maximum Rights Mask**.

# Obtain File or Subdirectory Information 0x2222 87 06

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (06)	byte
8	NameSpace	byte
9	DestNameSpace	byte
10	SearchAttributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The data values in NetWareInfoStruct, from the least significant byte to the most significant byte, are:

- ◆ First 5 bits indicate the day, from 1-31.
- ◆ Next 4 bits indicate the month, from 1-12.
- ◆ Last 7 bits indicate the year, with 0=1980 and 20=2000.

The time values in `NetWareInfoStruct`, from the least significant byte to the most significant byte, are:

- ♦ First 5 bits indicate the number of 2-second intervals, from 0-29 so that 59 and 60 seconds are both indicated by 29.
- ♦ Next 6 bits indicate the minute, from 0-59.
- ♦ Last 57 bits indicate the hour, from 0-23.

*SearchAttributes* is currently not used. Regardless of what value is passed in to this field, information on the file or subdirectory will be returned (if the file or subdirectory exists).

*NWHandlePathStruct* is of type `NetWareHandlePathStruct` (page 609).

# Open CallBack Control 0x2222 87 34

NetWare Servers: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (34)	byte
8	CCFileHandle	long (Hi-Lo)
12	CCFunction	byte

## Parameters

### *CCFileHandle*

(Request) Specifies the file handle of the file being opened or created.

### *CCFunction*

(Request) Specifies one of three actions to be performed:

- 0x1 Clear Callback
- 0x2 Acknowledge Callback
- 0x3 Decline Callback
- 0x4 Level 2 OpLocks Enable

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The left most or most significant four bytes of *CCFileHandle* must be in Hi-Lo order.

# Open/Create File (old) 0x2222 84

Creates a new file for the calling client in the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (84)	byte
7	DirectoryHandle	byte
8	FileAttributes	byte
9	AccessFlags	byte
10	ActionCode	byte
11	FileNameLen	byte
12	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[6] (Hi-Lo)
14	Reserved	word (Hi-Lo)
16	FileName	byte[14]
30	FileAttributes	byte
31	FileExecuteType	byte
32	FileLen	long (Hi-Lo)
36	CreationDate	word (Hi-Lo)
38	LastAccessDate	word (Hi-Lo)
40	LastUpdateDate	word (Hi-Lo)
42	LastUpdateTime	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
132	0x84	No Create Privileges
133	0x85	No Create/Delete Privileges
135	0x87	Create Filename Error
141	0x8D	Some Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

**Create File 0x2222 67** creates and opens a file, automatically replacing an older file by the same name if it exists. **Create New File 0x2222 77** fails if another file by the same name exists. **Open/Create File** provides the *ActionFlag* parameter, which allows the client to specify one of three different actions:

0x1 ACTION\_OPEN  
0x2 ACTION\_REPLACE  
0x10 ACTION\_CREATE

The client must have at least file creation privileges in the specified directory to successfully call **Open/Create File**.

If the client has file deletion privileges in the specified directory, a file with the same name already exists in the directory, and the client specified that the existing file is to be replaced, the existing file will be erased before the new file is created.

If the client does not have file deletion privileges in the indicated directory and file with the same name already exists in the directory, the request will fail.



The newly created file will be stamped with the date and time of its creation. The file attributes byte will be set to the attributes specified by the client. The file will be opened as an exclusive file with both read and write access requested. (The actual access rights that are granted depend on the client's file access privileges in the specified directory.)

*FileHandle* requires the left most or most significant four bytes in Hi-Lo order.

**Open/Create File** is replaced by **Open Create File or Subdirectory 0x2222 87 01**.

## See Also

**Create File 0x2222 67 (page 306), Create New File 0x2222 77 (page 308), Erase File 0x2222 68 (page 317), Open File 0x2222 76 (page 417), Open/Create File or Subdirectory 0x2222 87 01 (page 398)**

# Open/Create File or Subdirectory 0x2222 87 01

Creates a new file for the calling client in the specified directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (01)	byte
8	<b>NameSpace</b>	byte
9	<b>OpenCreateMode</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	CreateAttributes	long (Lo-Hi)
20	<b>DesiredAccessRights</b>	word (Lo-Hi)
22	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	long (Hi-Lo)
12	<b>OpenCreateAction</b>	byte
13	Reserved	byte
14	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Parameters

*NameSpace*

(Request) Specifies the name space (see “**NameSpace Values**” on page 636).

### *OpenCreateMode*

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### *SearchAttributes*

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

### *ReturnInfoMask*

(Request) Specifies the specific information you want returned about a file or subdirectory.

### *CreateAttributes*

(Request) Specifies the DOS name space attributes.

### *DesiredAccessRights*

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### *FileHandle*

(Reply) Specifies the file handle of the file being opened or created.

### *OpenCreateAction*

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### *NetWareInfoStruct*

(Reply) Points to the NetWareInformationStructure.

### *NetWareFileNameStruct*

(Reply) Points to the NetWareFileNameStruct, which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

In NetWare 4.11 or above, the OS/2 name space (OS2.NAM) has been replaced with the LONG name space (LONG.NAM).

Beginning with NetWare 4.1, a file can be opened as a temporary file that will be deleted when the file is closed. If the item being created is a subdirectory, *DesiredAccessRights* is used as an inherited rights filter. Usually, you want all rights in the inherited rights mask definition and would pass 0xFF.

To create a temporary file, you must set DELETE\_FILE\_CLOSE\_BIT (0x0400), DENY\_READ\_BIT, and DENY\_WRITE\_BIT in *DesiredAccessRights*. By definition, temporary

files do not allow shared I/O access. The newly created file will exist until you close the file, which will cause the file to be deleted.

The OS creates temporary files with the `HIDDEN_BIT` set so they will not be visible. Also, the `IMMEDIATE_COMPRESS_BIT` is masked off from the created attributes that you supply.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct](#) (page 609).

## See Also

[Create File 0x2222 67](#) (page 306), [Create New File 0x2222 77](#) (page 308), [Erase File 0x2222 68](#) (page 317), [Open File 0x2222 76](#) (page 417), [Open/Create File or Subdirectory 0x2222 87 01](#) (page 398)

# Open/Create File or Subdirectory 0x2222 87 30

Creates or opens the file (depending on *OpenCreateMode*).

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (30)	byte
8	<b>Namespace</b>	byte
9	<b>DataStream</b>	byte
10	<b>OpenCreateMode</b>	byte
11	Reserved	byte
12	<b>SearchAttributes</b>	word (Lo-Hi)
14	Reserved	word (Lo-Hi)
16	<b>ReturnInfoMask</b>	long (Lo-Hi)
20	CreateAttributes	long (Lo-Hi)
24	<b>DesiredAccessRights</b>	word (Lo-Hi)
26	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	<b>OpenCreateAction</b>	byte
13	Reserved	byte
14	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Parameters

### *Namespace*

(Request) Specifies the name space (see “[Namespace Values](#)” on page 636).

### *DataStream*

(Request) Specifies the data stream number if the name space is Macintosh (see “[DataStream Values](#)” on page 633).

### *OpenCreateMode*

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### *SearchAttributes*

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

### *ReturnInfoMask*

(Request) Specifies the specific information you want returned about a file or subdirectory.

### *CreateAttributes*

(Request) Specifies the DOS name space attributes.

### *DesiredAccessRights*

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### *FileHandle*

(Reply) Specifies the file handle of the file being opened or created.

### *OpenCreateAction*

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### *NetWareInfoStruct*

(Reply) Points to the NetWareInformationStructure.

### *NetWareFileNameStruct*

(Reply) Points to the NetWareFileNameStruct, which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory** replaces **Open/Create File or Subdirectory 0x2222 87 01**. Subdirectories can be created, but not opened, by the client.

In NetWare 4.11 or above, the OS/2 name space (OS2.NAM) has been replaced with the LONG name space (LONG.NAM).

Beginning with NetWare 4.1, a file can be opened as a temporary file that will be deleted when the file is closed. If the item being created is a subdirectory, the *DesiredAccessRights* is used as an inherited rights filter. Usually, you want all rights in the inherited rights mask definition and would pass 0xFF.

To create a temporary file, you must set DELETE\_FILE\_CLOSE\_BIT (0x0400), DENY\_READ\_BIT, and DENY\_WRITE\_BIT in *DesiredAccessRights*. By definition, temporary files do not allow shared I/O access. The newly created file will exist until you close the file, which will cause the file to be deleted.

The OS creates temporary files with the HIDDEN\_BIT set so they will not be visible. Also, the IMMEDIATE\_COMPRESS\_BIT is masked off from the created attributes that you supply.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct](#) (page 609).

## See Also

[Create File 0x2222 67 \(page 306\)](#), [Create New File 0x2222 77 \(page 308\)](#), [Erase File 0x2222 68 \(page 317\)](#), [Open File 0x2222 76 \(page 417\)](#), [Open/Create File or Subdirectory 0x2222 87 01 \(page 398\)](#)

# Open/Create File or Subdirectory with Callback 0x2222 87 32

Creates or opens the file and sets a callback flag.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (32)	byte
8	<b>Namespace</b>	byte
9	<b>OpenCreateMode</b>	byte
10	Search Attributes	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	CreateAttributes	long (Lo-Hi)
20	<b>DesiredAccessRights</b>	word (Lo-Hi)
22	NWHandlePathStruct (see <b>NetWareHandlePathStruct</b> (page 609))	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	<b>OpenCreateAction</b>	byte
13	OCRetFlags	byte
14	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Parameters

*Namespace*

(Request) Specifies the name space (see “**Namespace Values**” on page 636).



### *OpenCreateMode*

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### *SearchAttributes*

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

### *ReturnInfoMask*

(Request) Specifies the specific information you want returned about a file or subdirectory.

### *CreateAttributes*

(Request) Specifies the DOS name space attributes.

### *DesiredAccessRights*

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### *NWHandlePathStruct*

(Request) Specifies the [NetWareHandlePathStruct](#) (page 609), which is used to pass the file/subdirectory handle or path.

### *FileHandle*

(Reply) Specifies the file handle of the file being opened or created.

### *OpenCreateAction*

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### *OCRetFlags*

(Reply) Specifies a callback field that can contain two values:

- 1 Indicates that this request has been registered for callback if someone else tries to open the file
- 0 Indicates that no callback has been registered

### *NetWareInfoStruct*

(Reply) Points to the [NetWareInformationStructure](#), which contains information such as the creation date/time, attributes, inherited rights, name space, etc.

### *NetWareFileNameStruct*

(Reply) Points to the [NetWareFileNameStruct](#), which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory with Callback** is an enhancement of **Open/Create File or Subdirectory 0x2222 87 01**. On its return, the `OCRetFlags` (`OpenCallBackReturnFlags`) field in the reply structure notifies you of completion. Subdirectories may be created, but not opened, by the client.

In NetWare 4.11 or above, the OS/2 name space (`OS2.NAM`) has been replaced with the LONG name space (`LONG.NAM`).

Beginning with NetWare 4.1, a file can be opened as a temporary file that will be deleted when the file is closed. If the item being created is a subdirectory, the *DesiredAccessRights* is used as an inherited rights filter. Usually, you want all rights in the inherited rights mask definition and would pass `0xFF`.

To create a temporary file, you must set `DELETE_FILE_CLOSE_BIT` (`0x0400`), `DENY_READ_BIT`, and `DENY_WRITE_BIT` in *DesiredAccessRights*. By definition, temporary files do not allow shared I/O access. The newly created file will exist until you close the file, which will cause the file to be deleted.

The OS creates temporary files with the `HIDDEN_BIT` set so they will not be visible. Also, the `IMMEDIATE_COMPRESS_BIT` is masked off from the created attributes that you supply.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

## See Also

[Open/Create File or Subdirectory 0x2222 87 01 \(page 398\)](#)

# Open/Create File or Subdirectory II with Callback 0x2222 87 33

Creates or opens the file and sets a callback flag.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (33)	byte
8	Namespace	byte
9	DataStream	byte
10	OpenCreateMode	byte
11	Reserved	byte
12	Search Attributes	word (Lo-Hi)
14	Reserved	word (Lo-Hi)
16	ReturnInfoMask	long (Lo-Hi)
20	CreateAttributes	long (Lo-Hi)
24	DesiredAccessRights	word (Lo-Hi)
26	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	OpenCreateAction	byte
13	OCRetFlags	byte
14	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Parameters

### *Namespace*

(Request) Specifies the name space (see “[Namespace Values](#)” on page 636).

### *DataStream*

(Request) Specifies the data stream number if the name space is Macintosh (see “[DataStream Values](#)” on page 633).

### *OpenCreateMode*

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### *SearchAttributes*

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

### *ReturnInfoMask*

(Request) Specifies the specific information you want returned about a file or subdirectory.

### *CreateAttributes*

(Request) Specifies the DOS name space attributes.

### *DesiredAccessRights*

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### *FileHandle*

(Reply) Specifies the file handle of the file being opened or created.

### *OpenCreateAction*

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### *OCRetFlags*

(Reply) Specifies a callback field that can contain two values:

- 1 Indicates that this request has been registered for callback if someone else tries to open the file
- 0 Indicates that no callback has been registered

### *NetWareInfoStruct*

(Reply) Points to the NetWareInformationStructure.

### *NetWareFileNameStruct*

(Reply) Points to the NetWareFileNameStruct, which contains the name and length for the file or subdirectory.

# Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

# Remarks

**Open/Create File or Subdirectory II with Callback** is an enhancement of **Open/Create File or Subdirectory 0x2222 87 30**. On its return, the *OCRetFlags* (*OpenCallBackReturnFlags*) field in the reply structure notifies you of completion. Subdirectories may be created, but not opened, by the client.

In NetWare 4.11 or above, the OS/2 name space (OS2.NAM) has been replaced with the LONG name space (LONG.NAM).

Beginning with NetWare 4.1, a file can be opened as a temporary file that will be deleted when the file is closed. If the item being created is a subdirectory, the *DesiredAccessRights* is used as an inherited rights filter. Usually, you want all rights in the inherited rights mask definition and would pass 0xFF.

To create a temporary file, you must set *DELETE\_FILE\_CLOSE\_BIT* (0x0400), *DENY\_READ\_BIT*, and *DENY\_WRITE\_BIT* in *DesiredAccessRights*. By definition, temporary files do not allow shared I/O access. The newly created file will exist until you close the file, which will cause the file to be deleted.

The OS creates temporary files with the *HIDDEN\_BIT* set so they will not be visible. Also, the *IMMEDIATE\_COMPRESS\_BIT* is masked off from the created attributes that you supply.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

# See Also

[Open/Create File or Subdirectory 0x2222 87 30 \(page 401\)](#)

# Open/Create File or Subdirectory 0x2222 89 01

Creates or opens the specified file, depending on the OpenCreateMode parameter. Subdirectories can be created, but not opened, by the client.

**NetWare Servers:** 6.5, SP2 and later

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (01)	byte
8	<b>Namespace</b>	byte
9	<b>OpenCreateMode</b>	byte
10	Search Attributes	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	CreateAttributes	long (Lo-Hi)
20	<b>DesiredAccessRights</b>	word (Lo-Hi)
22	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	long (Hi-Lo)
12	<b>OpenCreateAction</b>	byte
13	Reserved	byte
14	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Parameters

*Namespace*

(Request) Specifies the name space (see “**Namespace Values**” on page 636).

### *OpenCreateMode*

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### *SearchAttributes*

(Request) Specifies the following create and open options (see “[SearchAttributes Values](#)” on page 639):

- ◆ creating or opening a hidden or system file
- ◆ creating a subdirectory only
- ◆ creating a subdirectory or creating or opening any normal file

### *ReturnInfoMask*

(Request) Specifies the specific information you are requesting about a file or subdirectory. Information such as the creation date and time, attributes, inherited rights, or the name space is returned in the NetWareInfoStruct structure.

### *CreateAttributes*

(Request) Specifies the DOS name space attributes.

### *DesiredAccessRights*

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### *FileHandle*

(Reply) Specifies the file handle of the file being opened or created.

### *OpenCreateAction*

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### *OCRetFlags*

(Reply) Specifies a callback field that can contain two values:

- 1 Indicates that this request has been registered for callback if someone else tries to open the file
- 0 Indicates that no callback has been registered

### *NetWareInfoStruct*

(Reply) Points to the NetWareInformationStructure.

### *NetWareFileNameStruct*

(Reply) Points to the NetWareFileNameStruct, which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory II with Callback** is an enhancement of **Open/Create File or Subdirectory 0x2222 87 30**. On its return, the `OCRetFlags` (`OpenCallBackReturnFlags`) field in the reply structure notifies you of completion. Subdirectories may be created, but not opened, by the client.

In NetWare 4.11 or above, the OS/2 name space (`OS2.NAM`) has been replaced with the LONG name space (`LONG.NAM`).

Beginning with NetWare 4.1, a file can be opened as a temporary file that will be deleted when the file is closed. If the item being created is a subdirectory, the *DesiredAccessRights* is used as an inherited rights filter. Usually, you want all rights in the inherited rights mask definition and would pass `0xFF`.

To create a temporary file, you must set `DELETE_FILE_CLOSE_BIT` (`0x0400`), `DENY_READ_BIT`, and `DENY_WRITE_BIT` in *DesiredAccessRights*. By definition, temporary files do not allow shared I/O access. The newly created file will exist until you close the file, which will cause the file to be deleted.

The OS creates temporary files with the `HIDDEN_BIT` set so they will not be visible. Also, the `IMMEDIATE_COMPRESS_BIT` is masked off from the created attributes that you supply.

*NWHandlePathStruct* is of type [NetWareHandlePathStruct \(page 609\)](#).

## See Also

[Open/Create File or Subdirectory 0x2222 87 30 \(page 401\)](#)



# Open Data Stream 0x2222 22 49

Opens a data stream associated with any supported name space on the server and returns a NetWare file handle.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (5 + FileNameLen)	word (Hi-Lo)
9	SubFunctionCode (49)	byte
10	DataStream	byte
11	DirHandle	byte
12	FileAttributes	byte
13	OpenRiughts (requested rights)	byte
14	FileNameLen	byte
15	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)

## Parameters

*DataStream*

(Request) Specifies the data stream number if the name space is Macintosh (see “**DataStream Values**” on page 633).

*DirHandle*

(Request) Specifies the directory handle of the directory in which the file is located.

*FileAttributes*

(Request) Specifies a 1-byte bit definition that controls access to the file (see “**FileAttributes Values**” on page 634).

### *OpenRights*

(Request) Specifies a 1-byte bit definition, of which the lower eight bits specify the requested access rights to the file (see “**OpenRights Values**” on page 637).

### *FileNameLen*

(Request) Specifies the length of the file name.

### *FileName*

(Request) Specifies the name of the file for which a data stream is opened.

### *FileHandle*

(Reply) Specifies the file handle returned by this request.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Failure
130	0x82	No Open Privileges
144	0x90	Read-only Access To Volume
190	0xBE	Invalid Data Stream
255	0xFF	Failure

## Remarks

*FileHandle* requires the left most or most significant 4 bytes in Hi-Lo order.

# Open File (old) 0x2222 65

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (65)	byte
7	DirectoryHandle	byte
8	SearchAttributes	byte
9	FileNameLen	byte
10	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[6] (Hi-Lo)
14	Reserved	word (Hi-Lo)
16	FileName	byte[14]
30	FileAttributes	byte
31	FileExecuteType	byte
32	FileLen	long (Hi-Lo)
36	CreationDate	word (Hi-Lo)
38	LastAccessDate	word (Hi-Lo)
40	LastUpdateDate	word (Hi-Lo)
42	LastUpdateTime	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
128	0x80	Lock Fail
129	0x81	Out Of Handles
130	0x81	No Open Privileges
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Lock Error, Failure, No Files Found

## Remarks

**Open File (old)** is the pre-NetWare 2.0a form of **Open File 0x2222 76** and is equivalent to **Open File**, except that *DesiredAccessRights* cannot be specified.

Calling **Open File (old)** is the same as calling **Open File 0x2222 76** with *DesiredAccessRights* set to Exclusive, Read, and Write (0x13).

*FileHandle* requires the left most or most significant four bytes in Hi-Lo order.

## See Also

**Open File 0x2222 76 (page 417), Close File 0x2222 66 (page 299), Open/Create File (old) 0x2222 84 (page 395), Open/Create File or Subdirectory 0x2222 87 01 (page 398)**

# Open File 0x2222 76

Opens an existing file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (76)	byte
7	DirectoryHandle	byte
8	SearchAttributes	byte
9	DesiredAccessRights	byte
10	FileNameLen	byte
11	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[6] (Hi-Lo)
14	Reserved	word (Hi-Lo)
16	FileName	byte[14]
30	FileAttributes	byte
31	FileExecuteType	byte
32	FileLen	long (Hi-Lo)
36	CreationDate	word (Hi-Lo)
38	LastAccessDate	word (Hi-Lo)
40	LastUpdateDate	word (Hi-Lo)
42	LastUpdateTime	word (Hi-Lo)

## Parameters

### *DesiredAccessRights*

(Request) Specifies which access rights a client wants in the specified file (see “**DesiredAccessRights Values**” on page 633).

### *FileName*

(Request) Specifies a valid file path and is used with *DirectoryHandle* to indicate which file should be opened.

### *FileHandle*

(Reply) Specifies the handle that must be used for all subsequent file access requests.

### *FileName*

(Reply) Specifies the name of the opened file (NULL-padded if fewer than 14 characters).

### *FileAttributes*

(Reply) Specifies the attributes of the opened file.

### *FileLength*

(Reply) Specifies the length of the file at the instant it was opened.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
130	0x81	No Open Privileges
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Lock Error, Failure, No Files Found

## Remarks

**Open File** is replaced by **Open Create File or Subdirectory** 0x2222 87 01.

If the client lacks file open privileges in the target directory, the request will fail.

If the file's shareable bit is cleared and the write bit is also cleared in *DesiredAccessRights*, the deny read bit is cleared and the deny write bit is set, which allows multiple clients to read from the file with no one writing to it.

If the file's shareable bit is cleared and the write bit is set in *DesiredAccessRights*, the deny read bit is set and the deny write bit is set, which ensures that the file is kept for the exclusive use of the requesting client.

The client's initial *DesiredAccessRights* are modified to reflect the actual access rights the client is allowed to the specified directory and file. After the flag has been modified, the result is compared with the access rights of other clients that are currently using the same file. If the *DesiredAccessRights* are not compatible with the access rights of other clients, the open request is refused.

The client must not modify any information within *FileHandle*. It requires that the left most or most significant 4 bytes are in Hi-Lo order.

If multiple clients are using a file, they must coordinate extending the shared file among themselves (see **Get Current Size of File 0x2222 71 (page 330)**).

Dates and times are in MS-DOS format, except that they are stored in Hi-Lo order.

Clients can open a single file more than once. However, the file must be closed as many times as it is opened before the server will release the file to other clients.

## See Also

**Open File (old) 0x2222 65 (page 415), Close File 0x2222 66 (page 299), Open/Create File (old) 0x2222 84 (page 395), Open/Create File or Subdirectory 0x2222 87 01 (page 398)**

# Parse Tree 0x2222 90 00

Parses a tree by a directory base, which allows you to query for information about files, directories, or the root.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStructLen (17 + limbStruct)	word (Hi-Lo)
9	SubFuncCode (00)	byte
10	InformationMask	long (Lo-Hi)
14	InformationMask2	long (Lo-Hi)
18	reserved	long (Lo-Hi)
22	limbCount	long (Lo-Hi)
26+	limbStruct	structure[]

## Reply Format

Offset	Content	Type
Reply header		
8	limbCompletedCnt	long (Lo-Hi)
12	ItemsCount	long (Lo-Hi)
16	nextLimbScanNum	long (Lo-Hi)
20+	InfoBlock	structure

## Parameters

*InformationMask2*

(Request) Is reserved for future use.

*limbCount*

(Request) Specifies the count of limb structures that are present in the request packet.



### *limbStructure*

(Request) Points to the limb structure, which contains information about an area to scan or return information about, depending on whether *ScanFolder* is set.

### *limbCompletedCnt*

(Reply) Specifies the number of limb structures that were processed and completed from the request packet.

### *ItemsCount*

(Reply) Specifies the number of *InfoBlock* structures.

### *nextLimbScanNum*

(Reply) Specifies the next scan value for the limb structure to be scanned:

-1 Scan has completed

### *InfoBlock*

(Reply) Points to the InfoBlock structure, which contains the requested information for each scanned item.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Lock Error, Failure, No Files Found

## Remarks

Multiple requests can be placed in the request packet to reduce traffic over the wire.

*nextLimbScanNum* is valid only if the Scan Entire Folder bit is set.

# Purge Erased Files (old) 0x2222 22 16

Permanently deletes all files that the file server is holding for the client on all the server's volumes.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStructLen (1)	word (Hi-Lo)
9	SubFunctionCode (16)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
129	0x81	Out Of Handles
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
161	0xA1	Directory I/O Error
255	0xFF	Failure

## Remarks

**Purge Erased Files** is replaced by **Purge Salvageable File 0x2222 22 29**.

When files are deleted by a client, they are moved to a holding area on the volume until they are either purged, restored (by calling **Restore Erased File 0x2222 22 17**), or replaced by other files that are deleted by the client.

The space relinquished by purged files can now be used by the file server. Deleted files cannot be recovered by calling **Restore Erased File**. Files that are deleted by the client after **Purge Erased Files** is called will again be placed in the holding area.

## See Also

**Recover Erased File (old) 0x2222 22 17 (page 433), Purge Salvageable File (old) 0x2222 22 29 (page 424), Recover Salvageable File (old) 0x2222 22 28 (page 436)**

# Purge Salvageable File 0x2222 87 18

Permanently deletes entries found by calling **Scan Salvageable Files**.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (18)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	ScanSequence	long (Lo-Hi)
14	ScanVolume	long (Lo-Hi)
18	ScanDirectoryBase	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

You cannot use file names or wildcards to search for search for salvageable files or subdirectories.

*ScanSequence* is the same value that was used in the **Scan Salvageable Files 0x2222 22 27** request.

## See Also

**Purge Salvageable File (old) 0x2222 22 29 (page 424), Recover Salvageable File (old) 0x2222 22 28 (page 436), Scan Salvageable Files (old) 0x2222 22 27 (page 474)**

# Purge Salvageable File (old) 0x2222 22 29

Permanently deletes entries found by calling **Scan Salvageable Files**.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (29)	byte
10	DirectoryHandle	byte
11	Sequence	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
133	0x85	No Delete Privileges
156	0x9C	Invalid Path

## Remarks

**Purge Salvageable File (old)** is replaced by **Purge Salvageable File 0x2222 87 18**.

## See Also

**Purge Salvageable File 0x2222 87 18 (page 423)**, **Recover Salvageable File (old) 0x2222 22 28 (page 436)**, **Scan Salvageable Files (old) 0x2222 22 27 (page 474)**

# Purge Salvageable File List 0x2222 87 42

Permanently deletes all entires in a subdirectory (or those found by calling **Scan Salvageable File List 0x2222 87 41**).

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (42)	byte
8	NameSpace	byte
9	reserved	byte
10	ControlFlags	word (Lo-Hi)
12	Volume	long (Lo-Hi)
16	PurgeBase	long (Lo-Hi)
20	PurgeCount	long (Lo-Hi)
24	PurgeList	long[] (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	PurgeCount	long (Lo-Hi)
12	PurgeListCcode	long[] (Lo-Hi)

## Parameters

### *ControlFlags*

(Request) Specifies whether to purge all files:

0x00000001   PurgeAll

### *Volume*

(Request) Specifies the number of the volume on which the items to be purged reside.

### *PurgeBase*

(Request) Specifies the NetWare directory base of the path to be purged as returned from **Scan Salvageable File List 0x2222 87 41**.

### *PurgeCount*

(Request) Specifies the number of items in the list to be purged (if *PurgeAll* is not set).

### *PurgeList*

(Request) Specifies a list of salvageable file entry numbers obtained by calling **Scan Salvageable File List 0x2222 87 41**.

### *PurgeCount*

(Reply) Specifies the number of *PurgeListCcode* parameters that are returned in the list.

### *PurgeListCcode*

(Reply) Specifies the corresponding ccode for each item in *PurgeList*.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If *PurgeAll* was set in *ControlFlags*, *PurgeCount* is ignored and no items need to be placed in *PurgeList*. All salvageable files in the subdirectory specified by *PurgeBase* will be purged.

If *PurgeAll* was set, *PurgeCount* contains 1, indicating there is only 1 valid field in *PurgeListCcode*; and *PurgeListCcode* will contain only 1 entry, indicating the total number of salvageable files that were purged.

If you obtained a list of salvageable files from **Scan Salvageable File List 0x2222 87 41**, the list must contain only salvageable file entry numbers and no file names.

## See Also

**Purge Salvageable File (old) 0x2222 22 29 (page 424), Recover Salvageable File (old) 0x2222 22 28 (page 436), Scan Salvageable Files (old) 0x2222 22 27 (page 474), Scan Salvageable File List 0x2222 87 41 (page 470)**

# Query NS Information Format 0x2222 87 23

Queries for the format of name space information.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (23)	byte
8	<b>NameSpace</b>	byte
9	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	FixedBitMask	long (Lo-Hi)
12	VariableBitMask	long (Lo-Hi)
16	HugeBitMask	long (Lo-Hi)
20	FixedBitsDefined	word (Lo-Hi)
22	VariableBitsDefined	word (Lo-Hi)
24	HugeBitsDefined	word (Lo-Hi)
26	FieldsLenTable[32]	long (Lo-Hi)

## Parameters

### *NameSpace*

(Request) Specifies one of the following name spaces (see “**NameSpace Values**” on [page 636](#)).

### *Volume*

(Request) Specifies the volume for which name space information is requested.

### *FieldsLengthTable*

(Reply) Specifies the length of the information relative to any of the masks.

# Return Values

Decimal	Hex	Description
0	0x00	Successful

# Remarks

In NetWare 4.11 or above, the OS/2 name space (OS2.NAM) has been replaced with the LONG name space (LONG.NAM).

Only one of the three masks (*FixedBitMask*, *VariableBitMask*, and *HugeBitMask*) can set a specific bit position. The name space information bit mask is produced by ORing information from all three masks (see “**Name Space Information Bit Mask**” on page 283).

For example, the server might return the following information for the DOS name space:

```
FixedBitMask: 0. . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . 0
VariableBitMask: 0. . . 0 0 0 0 0 0 0 0 0 0 0 0 . . . 0
HugeBitMask: 0. . . 0 0 0 0 0 0 0 0 0 0 0 0 . . . 0
```

DOS Name Space *FieldsLengthTable* Definition (BYTES[]): 12, 4, 2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

DOS Name Space NSInfoBitMask would be:

- 0    Modify Name Bit
- 1    File Attributes Bit
- 2    Create Date Bit
- 3    Create Time Bit
- 4    Owner ID Bit
- 5    Archive Date Bit
- 6    Archive Time Bit
- 7    Archive ID Bit
- 8    Modify Date Bit
- 9    Modify Time Bit
- 10    Modify ID Bit
- 11    Last Accessed Date Bit
- 12    Inheritance Bit
- 13    Maximum Space Bit
- 14-31    reserved

# See Also

**Get NS Information 0x2222 87 19 (page 357), Set NS Information 0x2222 87 25 (page 504)**



# Read File 0x2222 87 64

Reads a block of bytes from a file starting at the specified file offset.

NetWare Servers: 6.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (64)	byte
8	FileHandle	long (Hi-Lo)
12	StartingByteOffset	UINT64 (Hi-Lo)
20	BytesToRead	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	BytesActuallyRead	word (Hi-Lo)
10	Data	byte[BytesActuallyRead]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
253	0xFD	Bad Station Number
255	0xFF	I/O Bound Error

## Remarks

Note that *StartingByteOffset* in the request can be 64 bits. Requests made to NSS volumes support reading at true 64-bit offsets. Requests made to traditional volumes support reading at 32-bit offsets. Reading at offsets greater than 32 bits on a traditional volume result in an error.

If the EOF is encountered before the read request is satisfied, the server returns Successful, but *BytesActuallyRead* contains a count less than *BytesToRead*.

If the calling client does not have read access privileges to the indicated file, or if a portion of the targeted byte block is locked for use by some other client, **Read File** will fail.

Clients are constrained by the current negotiated file buffer size (see **Negotiate Buffer Size** 0x2222 33). A client cannot request more bytes of data than will fit in the currently negotiated buffer size.

For NetWare versions prior to 5.0, the client cannot request a data block that straddles a buffer-size boundary in the file. For example, if the current buffer size were 512 bytes and the client wants to read 1000 bytes starting at file offset 500, the client must issue three read requests. The first request will read 12 bytes starting at offset 500. The second request will read 512 bytes starting at offset 512. The last request will read 476 bytes starting at offset 1024.

If a client requests a block of data that starts on an odd-byte boundary within the file, the first byte of the data field returned by the server will contain garbage. The actual data from the file will start in the second byte of the data block.

*FileHandle*, *StartingByteOffset*, and *BytesToRead* require the input to be in Hi-Lo order. *BytesActuallyRead* is returned in Hi-Lo order.

## See Also

[Write to a File 0x2222 87 65 \(page 507\)](#)

# Read From a File 0x2222 72

Reads a block of bytes from a file starting at a specified file offset.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (72)	byte
7	Reserved	byte
8	FileHandle	byte[6] (Hi-Lo)
14	StartingByteOffset	long (Hi-Lo)
18	BytesToRead	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	BytesActuallyRead	word (Hi-Lo)
10	Data	byte[BytesActuallyRead]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
147	0x93	No Read Privileges
255	0xFF	I/O Bound Error

## Remarks

If the EOF is encountered before the read request is satisfied, the server returns Successful, but *BytesActuallyRead* will contain a count less than *BytesToRead*.

If the calling client does not have read access privileges to the indicated file, or if a portion of the targeted byte block is locked for use by some other client, **Read From a File** will fail.

Clients are constrained by the current negotiated file buffer size (see **Negotiate Buffer Size** 0x2222 33). A client cannot request more bytes of data than will fit in the currently negotiated buffer size.

For NetWare versions prior to 5.0, the client cannot request a data block that straddles a buffer-size boundary in the file. For example, if the current buffer size were 512 bytes and the client wants to read 1000 bytes starting at file offset 500, the client must issue three read requests. The first request will read 12 bytes starting at offset 500. The second request will read 512 bytes starting at offset 512. The last request will read 476 bytes starting at offset 1024.

For NetWare 5.0 and later, the client can request a data block that straddles a buffer-size boundary in the file. However, the client is still constrained by the current negotiated file buffer size, and thus cannot request more bytes of data than will fit in the currently negotiated buffer size.

If a client requests a block of data that starts on an odd-byte boundary within the file, the first byte of the data field returned by the server will contain garbage. The actual data from the file will start in the second byte of the data block.

*FileHandle* requires the left most or most significant 4 bytes in Hi-Lo order.

# Recover Erased File (old) 0x2222 22 17

Recovers one file on the specified volume.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (3 + PathNameLen)	word (Hi-Lo)
9	SubFunctionCode (17)	byte
10	DirectoryHandle	byte

## Reply Format

Offset	Content	Type
Reply header		
8	OldFileName	byte[15]
23	NewFileName	byte[15]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Hard Failure, Failure

## Remarks

**Recover Erased File (old)** is replaced by **Recover Salvageable File 0x2222 22 28**.

When a client requests that a file be erased, the file is marked for deletion and moved to a special holding area in the volume's directory structure. Deleted files that are in the holding area can be recovered only until the client attempts another file erase or file create request. The server holds only the most recently deleted files for each client and purges the older files so that their disk and directory space can be reclaimed.

*OldFileName* must contain the file's NULL-padded, original name. If a file having the same name already exists in the target directory, the server will assign a new file name. The new file name will be recorded in *NewFileName* and will also be NULL-padded (if necessary).

## See Also

**Purge Erased Files (old) 0x2222 22 16 (page 422), Recover Salvageable File (old) 0x2222 22 28 (page 436), Purge Salvageable File (old) 0x2222 22 29 (page 424)**

# Recover Salvageable File 0x2222 87 17

Recovers a file or subdirectory entry that was found by calling **Scan Salvageable Files**.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (17)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	ScanSequence	long (Lo-Hi)
14	ScanVolume	long (Lo-Hi)
18	ScanDirectoryBase	long (Lo-Hi)
22	NewFileNameLen	byte
23	NewFileName	byte[NewFileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

You cannot use file names or wildcard characters to search for salvageable files or subdirectories.

*ScanSequence* is the value that was used in the **Scan Salvageable Files** request.

# Recover Salvageable File (old) 0x2222 22 28

Recovers the specified salvageable file into the same directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen	word (Lo-Hi)
9	SubFunctionCode (28)	byte
10	DirectoryHandle	byte
11	Sequence	long (Hi-Lo)
15	FileNameLen	byte
16	FileName	byte[FileNameLen]
16 + FileNameLen	newFileNameLen	byte
17 + FileNameLen	newFileName	byte[newFileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
132	0x84	No Create Privileges
156	0x9C	Invalid Path
254	0xFE	File Name Already Exists In This Directory

## Remarks

**Recover Salvageable File (old)** is replaced by **Recover Salvageable File 0x2222 87 17**.

$SubFuncStrucLen = 8 + FileNameLen + newFileNameLen$

## See Also

**Purge Salvageable File (old) 0x2222 22 29 (page 424), Scan Salvageable Files (old) 0x2222 22 27 (page 474), Recover Salvageable File 0x2222 87 17 (page 435)**



# Remove Extended Trustee from Dir or File 0x2222 22 43

Removes a trustee from a file or directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (5 + PathLen)	word (Lo-Hi)
9	SubFunctionCode (43)	byte
10	DirHandle	byte
11	ObjectID	long (Hi-Lo)
15	Unused	byte
16	PathLen	byte
17	Path	byte[PathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
144	0x90	Read-only Access To Volume
156	0x9C	Invalid Path
254	0xFE	Trustee Not Found
255	0xFF	No Trustee Change Privileges

## See Also

**Add Extended Trustee to Directory or File 0x2222 22 39 (page 286)**

# Remove User Disk Space Restrictions 0x2222 22 34

Removes a user's disk space restriction from the specified volume.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Lo-Hi)
9	SubFunctionCode (34)	byte
10	VolumeNumber	byte
11	ObjectID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
254	0xFE	User Not Found

## See Also

**Get Directory Disk Space Restriction 0x2222 22 35 (page 332), Add User Disk Space Restriction 0x2222 22 33 (page 290), Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366), Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476), Set Directory Disk Space Restriction 0x2222 22 36 (page 487), Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Rename Directory 0x2222 22 15

Renames a directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen	word (Lo-Hi)
9	SubFunctionCode (15)	byte
10	DirectoryHandle	byte
11	DirectoryPathLen	byte
12	DirectoryPath	byte[DirectoryPathLen]
12 + DirectoryPathLen	NewDirectoryPathLen	byte
13 + DirectoryPathLen	NewDirectoryPath	byte[NewDirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
139	0x8B	No Rename Privileges
146	0x92	All Names Exist
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
158	0x9E	Bad File Name
161	0xA1	Directory I/O Error
239	0xEF	Illegal Name
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

*NewDirectoryPath* must contain only the new name of the target directory and must not be preceeded by a path specification. The current implementation limits directory names to the DOS 8.3 naming conventions. Longer paths will be truncated.

The client must have access control and modify rights in the directory to call **Rename Directory**.

$SubFuncStructLen = 4 + DirectoryPathLen + NewDirectoryPathLen$

## See Also

**Rename Or Move (old) 0x2222 22 46 (page 443), Rename Or Move a File or Subdirectory 0x2222 87 04 (page 445)**

# Rename File 0x2222 69

Renames a file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (69)	byte
7	DirectoryHandle	byte
8		byte
9	FileNameLen	byte
10	FileName	byte[FileNameLen]
10 + FileNameLen	TargetDirectoryHandle	byte
11 + FileNameLen	NewFileNameLen	byte
12 + FileNameLen	NewFileName	byte[NewFileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
135	0x87	Create Filename Error
139	0x8B	No Rename Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
145	0x91	Some Names Exist
146	0x92	All Names Exist
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
154	0x9A	Rename Across Volume

Decimal	Hex	Description
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

**Rename File** is replaced by **Rename or Move a File or Subdirectory 0x2222 87 04**.

The source directory (where the file resides) and the target directory (where the renamed file will be located) do not need to be the same directory. **Rename File** can be called to move a file from one directory to another. However, the two directories must reside on the same server volume.

The client must have file modification privileges in both the source and the target directories. If the file is being used by other clients or if the target name already exists in the target directory, the rename request will fail.

You can use wildcard characters to rename a file.

## See Also

**Rename Or Move a File or Subdirectory 0x2222 87 04 (page 445)**

# Rename Or Move (old) 0x2222 22 46

Renames a file or directory by using long names.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (46)	byte
10	SourceDirHandle	byte
11	SearchAttribute	byte
12	SourcePathComponentCount	byte
13	SourcePath	byte[SourcePathLen]
13 + SourcePathLen	DestDirHandle	byte
14 + SourcePathLen	DestPathComponentCount	byte
15 + SourcePathLen	DestPath	byte[DestPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Insufficient Disk Space
135	0x87	Create Filename Error
139	0x8B	No Rename Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
145	0x91	Some Names Exist
146	0x92	All Names Exist

Decimal	Hex	Description
154	0x9A	Rename Across Volume
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
164	0xA4	Invalid Directory Rename Attempted
255	0xFF	No Files Found

## Remarks

**Rename Or Move (old)** supports wildcard characters. Because you pass it a full path, **Rename Or Move (old)** can be used to move files and directories by changing the target path.

Unlike most strings, which are preceeded by their lengths, the source and destination paths are preceded by the number of components in the path strings. For example, the path "TEST\TESTDIR" consists of two components. Each entry in the path is preceded by its length so that "TEST\TESTDIR" will be "\X004TEST\X007TESTDIR."

**Rename Or Move (old)** will move entries within the same volume only.

## See Also

**Rename Or Move a File or Subdirectory 0x2222 87 04 (page 445), Rename Directory 0x2222 22 15 (page 439)**



# Rename Or Move a File or Subdirectory 0x2222 87 04

NetWare Servers: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (04)	byte
8	NameSpace	byte
9	RenameFlag	byte
10	SearchAttributes	word (Lo-Hi)
12	SrcNWHandlePathS1	structure
19	DstNWHandlePathS2	structure
26	SrcDstPathStrings[]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*SrcNWHandlePathS1* and *DstNWHandlePath2* are of type *NWHandlePathStruct*, except that, due to space limitations, the *PathInfo* array is not contained in these structures. *ComponentCount* in these two structures is used to specify how many components exist for each structure.

*SrcDstPathStrings* contains the source component path (if present) and the destination component path (if present).

# Restore an Extracted Base Handle 0x2222 22 24

Obtains a directory handle from the NetWare internal identifier for the specified directory handle ID.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (15 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (24)	byte
10	ServerNetworkAddress	byte[10]
20	HandleID	byte[4]

## Reply Format

Offset	Content	Type
Reply header		
8	NewDirectoryHandle	byte
9	AccessRightsMask	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
157	0x9D	No Directory Handles
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number

Decimal	Hex	Description
255	0xFF	Failure

## Remarks

*HandleID* must be obtained by calling **Extract a Base Handle 0x2222 22 23**.

## See Also

**Restore an Extracted Base Handle 0x2222 22 24 (page 446)**

# Revoke File Handle Rights 0x2222 87 43

Revokes the rights for the specified file handle.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (43)	byte
8	Reserved[3]	byte
11	QueryFlag	byte
12	FileHandle	long (Hi-Lo)
16	RemoveOpenRights	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	long (Hi-Lo)
12	ResultantRights	long (Lo-Hi)

## Parameters

### *Query Flag*

(Request) Specifies whether to query the locks engine and return the current effective access rights of the file handle:

0x1 Query the locks engine and return the access rights

### *FileHandle*

(Request) Specifies the file handle of the file being opened or created.

### *RemoveOpenRights*

(Request) Specifies the original access rights mask under which the file was opened.

### *ResultantRights*

(Reply) Specifies the current effective access rights on the open file specified by *FileHandle*.

# Return Values

Decimal	Hex	Description
0	0x00	Successful
115	0x73	Revoke Handle Rights Not Found
136	0x88	Invalid File Handle
253	0xFD	Bad Station Number: File Is Not Opened Multiple Times By Same Station & Task

# Remarks

NetWare always assumed that a client would open a file only once when using the same connection number (station) and task number. If a client wanted to open the same file again, we assumed that the client would use a different task number. However, Windows 95 and Windows NT clients can open the same file multiple times using the same connection number and task number.

NetWare currently works as long as the requested rights (READ, WRITE, DENY\_READ, DENY\_WRITE) start from more rights to less rights. For example, open a file with READ and WRITE rights, then open the file again with READ, WRITE, DENY\_READ, and DENY\_WRITE. After closing the file (the second open occurrence), the rights of the open file are still set to READ, WRITE, DENY\_READ. Notice that the rights do not revert to READ and WRITE.

If a file is opened multiple times with the same connection and task numbers, the client can change the access rights back to what they were before the corresponding open had occurred by calling **Revoke File Handle Rights**.

The left most or most significant four bytes of *FileHandle* must be in Hi-Lo order.

*RemoveOpenRights* is used only if *QueryFlag* is 0x0.

# See Also

**Update File Handle Rights 0x2222 87 44 (page 450)**

# Update File Handle Rights 0x2222 87 44

Updates the rights for the specified file handle.

**NetWare Servers:** 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (44)	byte
8	Reserved[2]	byte
10	VolumeNumber	byte
11	NameSpace	byte
12	DirectoryNumber	long (Lo-Hi)
16	OldAccessRights	word (Lo-Hi)
18	NewAccessRights	word (Lo-Hi)
20	FileHandle	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	long (Hi-Lo)
12	ResultantRights	long (Lo-Hi)

## Parameters

*VolumeNumber*

(Request) Specifies the number of the volume the *DirectoryNumber* is on.

*NameSpace*

(Request) Specifies the name space to which the *DirectoryNumber* corresponds.

*OldAccessRights*

(Request) Used to verify that the old rights correspond with the existing rights.

### *NewAccessRights*

(Request) Specifies the value passed in to which the existing file handle's rights are to be updated.

### *FileHandle*

(Request) The file handle of the file currently open. The left-most or most significant four bytes must be in Hi-Lo order.

### *FileHandle*

(Reply) Specifies the file handle of the file currently open and just updated.

### *ResultantRights*

(Reply) Specifies the current effective access rights on the open file specified by *FileHandle*, after applying the *NewAccessRights*.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
115	0x73	Revoke Handle Rights Not Found
136	0x88	Invalid File Handle
253	0xFD	Bad Station Number: File Is Not Opened Multiple Times By Same Station & Task

## Remarks

NetWare originally assumed that a client would open a file only once when using the same connection number (station) and task number. If a client wanted to open the same file again, it was assumed that the client would use a different task number. However, Windows 95 and Windows NT clients can open the same file multiple times using the same connection number and task number.

NetWare currently works correctly as long as the requested rights (READ, WRITE, DENY\_READ, DENY\_WRITE) start from more rights to less rights. For example, open a file with READ and WRITE rights, then open the file again with READ, WRITE, DENY\_READ, and DENY\_WRITE. After closing the file (the second open occurrence), the rights of the open file are still set to READ, WRITE, DENY\_READ. Notice that the rights do not revert to READ and WRITE. Using this NCP handles this situation correctly.

## See Also

**Revoke File Handle Rights 0x2222 87 43 (page 448)**

# Scan a Directory 0x2222 22 30

Scans a directory using an 8.3 wildcard (server wildcard format) and returns information about the file or directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (8 + SearchPatternLen)	word (Hi-Lo)
9	SubFunctionCode (30)	byte
10	DirectoryHandle	byte
11	<b>SearchAttributes</b> (DOS file attributes)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Sequence	long (Lo-Hi)
12	Subdirectory	long
16	Attributes	long
20	UniqueID	byte
21	Flags	byte
22	Namespace	byte

## Reply Format (DOS File Entry)

Offset	Content	Type
23	NameLen	byte
24	Name	byte[12]
36	CreationDateAndTime	long (Lo-Hi)
40	OwnerID	long (Hi-Lo)



Offset	Content	Type
44	LastArchivedDateAndTime	long (Lo-Hi)
48	LastArchivedID	long (Hi-Lo)
52	LastUpdatedDateAndTime	long (Lo-Hi)
56	LastUpdatedID	long (Hi-Lo)
60	FileSize	long (Hi-Lo)
64	Reserved	byte[44]
108	InheritedRightsMask	word (Lo-Hi)
110	LastAccessedDate	word (Lo-Hi)
112	Reserved	byte[28]

## Reply Format (DOS Subdirectory Entry)

Offset	Content	Type
Reply header		
23	DirectoryNameLen	byte
24	DirectoryName	byte[12]
36	CreateDateAndTime	long (Lo-Hi)
40	OwnerID	long (Hi-Lo)
44	LastArchivedDateAndTime	long (Lo-Hi)
48	LastArchivedID	long (Hi-Lo)
52	LastModifiedDateAndTime	long (Lo-Hi)
56	NextTrusteeEntry	long (Hi-Lo)
60	Reserved	byte[48]
108	MaximumSpace	long (Lo-Hi)
112	InheritedRightsMask	word (Lo-Hi)
114	Undefined	byte[26]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	386 File Structure Not Supported On Server
255	0xFF	Failure

## Remarks

To initialize a scan, set *Sequence* to 0xFFFFFFFFh. To continue scanning, pass the previously returned sequence number.

*DirectoryHandle* cannot be a nonzero value; it must contain a directory handle.

*SearchPattern* must include a file name or wildcards. It cannot contain a path name.

## See Also

**Scan Directory Disk Space 0x2222 22 40 (page 455)**

# Scan Directory Disk Space 0x2222 22 40

Scans a directory using the wildcard format.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (8 + SearchPatternLen)	word (Hi-Lo)
9	SubFunctionCode (40)	byte
10	DirectoryHandle	byte
11	SearchAttributes	byte
12	Sequence	long (Lo-Hi)
16	SearchPatternLen	byte
17	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	Sequence	long (Lo-Hi)
12	Subdirectory	long (Lo-Hi)
16	Attributes	long (Lo-Hi)
20	UniqueID	byte
21	Flags	byte
22	NameSpace	byte
23	NameLen	byte
24	Name	byte[12]
36	CreationDateAndTime	long (Lo-Hi)
40	OwnerID	long (Hi-Lo)
44	LastArchivedDateAndTime	long (Lo-Hi)

Offset	Content	Type
48	LastArchivedID	long (Hi-Lo)
52	LastUpdatedDateAndTime	long (Lo-Hi)
56	LastUpdatedID	long (Hi-Lo)
60	DataForkSize	long (Hi-Lo)
64	DataForkFirstFAT	long (Hi-Lo)
68	NextTrusteeEntry	long (Hi-Lo)
72	Reserved	byte[36]
108	InheritedRightsMask	word (Lo-Hi)
110	LastAccessedDate	word (Lo-Hi)
112	DeletedFileTime	long (Lo-Hi)
116	DeletedDateAndTime	long (Lo-Hi)
120	DeletedID	long (Hi-Lo)
124	Undefined	byte[8]
132	PrimaryEntry	long (Lo-Hi)
136	NameList	long (Lo-Hi)
140	OtherFileForkSize	long[2] (Hi-Lo)

## Parameters

### *DataForkFirstFAT*

Specifies the first file allocation table (FAT) entry for the indicated file.

### *OtherFileForkSize*

Specifies two longs, which specify the file size for the data stream that is supported by the given name space and the first FAT entry for the name space-specific data stream respectively.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
137	0x89	No Search Privileges
156	0x9C	Invalid Path
251	0xFB	386 File Structure Not Supported On Server
255	0xFF	Failure

## Remarks

To initialize a directory search, set *Sequence* to 0xFFFFFFFFh.

**Scan Directory Disk Space** is similar to **Scan a Directory 0x2222 22 30**, except that this request also returns the data fork and resource fork size.

The file size is the actual file size, rather than the logical file size (sparse files can logically be much larger than they actually are).

## See Also

**Scan a Directory 0x2222 22 30 (page 452)**

# Scan Directory for Trustees 0x2222 22 12

Returns the objects that are trustees of the specified directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (12)	byte
10	DirectoryHandle	byte
11	TrusteeSetNumber	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryPath	byte[16]
24	CreationDate	word (Hi-Lo)
26	CreationTime	word (Hi-Lo)
28	DirectoryOwnerID	long (Hi-Lo)
32	TrusteeIDSet	long[5] (Hi-Lo)
52	TrusteeAccessMask	byte[5]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory

Decimal	Hex	Description
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

**Scan Directory for Trustees** must be called iteratively to retrieve all the trustees of a given directory. For the first request, set *TrusteeSetNumber* to 1; otherwise, unexpected results can occur. On subsequent requests, you must increment *TrusteeSetNumber* by 1. The server returns Failure when you request a *TrusteeSetNumber* that has no trustees.

*DirectoryPath* is NULL-padded.

*CreationDate* and *CreationTime* are in DOS format, except that the values are stored in Hi-Lo order.

A directory can have an arbitrary number of objects (usually users or user groups) listed as trustees. Each trustee has a corresponding *TrusteeAccessMasks*.

If a trustee is deleted, the deleted trustee ID number is replaced with a long zero, which indicates that the trustee slot is unused. Within each set of five trustee ID numbers, all zero entries are collected at the end of the trustee ID vector. Therefore, a client scanning a directory's trustee algorithm will stop scanning the current set of five trustees when it encounters a trustee ID of zero. The algorithm must then retrieve the next trustee set from the server and scan it. This process is repeated until the client receives a completion code other than Successful.

You must have access control rights to the target directory or to its parent directory to call **Scan Directory for Trustees**.

## See Also

**Scan File or Directory for Extended Trustees 0x2222 22 38 (page 466), Scan File or Subdirectory for Trustees 0x2222 87 05 (page 468)**

# Scan Directory Information 0x2222 22 02

Returns information about a file server directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (5 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (2)	byte
10	DirectoryHandle	byte
11	StartingSearchNumber	word (Hi-Lo)
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryPath	byte[16]
24	CreationDate	word (Hi-Lo)
26	CreationTime	word (Hi-Lo)
28	OwnerTrusteeID	word (Hi-Lo)
32	AccessRightsMask	byte
33	Reserved	byte
34	NextSearchNumber	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory



Decimal	Hex	Description
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

*DirectoryPath* can contain wildcard characters in the last name of the path since it is used as a pattern against which directory names are matched. When you call **Scan Directory Information** to search iteratively for wildcard patterns, set *StartingSearchNumber* to 1 on the first request. For subsequent requests, set *StartingSearchNumber* to 1 plus the *NextSearchNumber* returned with the previous request. *StartingSearchNumber* and *NextSearchNumber* are stored in Hi-Lo order.

*DirectoryPath* is NULL-padded.

*CreationDate* and *CreationTime* are in DOS format, except that the values are stored in Hi-Lo order.

*OwnerTrusteeID* contains the trustee ID of the object (user) that originally created the directory.

*AccessRightsMask* is a bit field that contains the maximum access privileges.

Scan Directory Information does not have to be used for iterative searches. A full or partial directory path can be specified, and the directory information for only that directory will be returned. If *DirectoryPathLen* is set to zero, *DirectoryPath* can be omitted, and the directory information for the specified directory will be returned.

## See Also

**File Search Initialize 0x2222 62 (page 324), File Search Continue 0x2222 63 (page 321)**

# Scan File Information 0x2222 23 15

Returns a file's extended status information, which includes the trustee ID of the owner who created the file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (6 + FileNameLen)	word (Hi-Lo)
9	SubFunctionCode (15)	byte
10	LastSearchIndex	word (Lo-Hi)
12	DirectoryHandle	byte
13	SearchAttributes	byte
14	FileNameLen	byte
15	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchIndex	word (Lo-Hi)
10	FileName	byte[14]
24	FileAttributes	byte
25	ExtendedFileAttributes	byte
26	FileSize	long (Hi-Lo)
30	FileCreationDate	word (Hi-Lo)
32	LastAccessedDate	word (Hi-Lo)
34	LastUpdateDate	word (Hi-Lo)
36	LastUpdateTime	word (Hi-Lo)
38	FileOwnerID	long (Hi-Lo)
42	LastArchiveDate	word (Hi-Lo)

Offset	Content	Type
44	LastArchiveTime	word (Hi-Lo)
46	Reserved	byte[56]

## Parameters

### *DirectoryHandle*

(Request) Specifies an index number from 1-255.

### *FileNameLen*

(Request) Specifies the length of the file name.

### *FileName*

(Request) Specifies a valid file path relative to *DirectoryHandle* (up to 255 bytes).

### *FileAttributes*

(Reply) Specifies the file attributes of the specified file.

### *ExtendedFileAttributes*

(Reply) Specifies the extended attributes of the specified file.

### *FileSize*

(Reply) Specifies the size of the specified file in bytes.

### *FileCreationDate* and *LastAccessDate*

(Reply) Specify the year, month, and day of the creation date and last access date of the specified file

### *LastUpdateTime*, *LastUpdateDate*, *LastArchiveTime*, and *LastArchiveDate*

(Reply) Specifies the year, month, day, hour, minute, and second of the last time the file was updated or archived.

### *OwnerObjectID*

(Reply) Specifies the bindery object ID of the user who created the file.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
137	0x89	No Search Privileges
147	0x93	No Read Privileges
148	0x94	No Write Privileges

Decimal	Hex	Description
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	No Files Found

## Remarks

You must have directory search privileges in the target directory to call **Scan File Information**.

**Scan File Information** is similar to **Search for a File** 0x2222 64. You should initially set *LastSearchIndex* to -1. If this request is called iteratively to scan entire in a directory, each subsequent request should set *LastSearchIndex* to the *NextSearchIndex* returned from the previous request.

A file server maintains a directory handle table for each logged in workstation. A directory handle points to a volume or a directory on the file server.

A full file path appears in the following format:

volume:director/.../directory/filename

A partial file path includes a file name and (optionally) one or more antecedent directory names. A file name can be 1 to 8 characters and can also include an extension of 1 to 3 characters. All letters must be upper case. A partial file path can be combined with a directory handle to identify a file. When you pass a full file path, you should also pass a value of 0x00 in to *DirectoryHandle*.

Dates are returned in the following format:

Byte 1								Byte 2							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
year								month				day			
0 to 119 (1980-2099)								1 to 12				1 to 31			

Times are returned in the following format:

Byte 3								Byte 4							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
hour				minute				second							
0 to 23				0 to 59				0 to 29 (2 second intervals)							

## See Also

[Search for a File 0x2222 64 \(page 478\)](#), [Set File Attributes 0x2222 70 \(page 495\)](#)

# Scan File or Directory for Extended Trustees 0x2222 22 38

Returns the extended trustee information for a file or directory (up to 20 entries per request).

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (4 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (38)	byte
10	DirHandle	byte
11	Sequence	byte
12	PathLen	byte
13	Path	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfEntries	byte
9	ObjectID	long[20] (Hi-Lo)
89	TrusteeRights	word[20]

## Parameters

*DirHandle*

(Request) Specifies the directory handle for the file or directory being scanned.

*Sequence*

(Request) Specifies the path length of the file or directory being scanned.

*PathLen*

(Request) Specifies the length of the path of the file or directory being scanned.

*NumberOfEntries*

(Reply) Specifies how many *ObjectID* and *TrusteeRights* parameters are actually returned.

*ObjectID*

(Reply) Specifies the ID number of the file or directory object being scanned.

*TrusteeRights*

(Reply) Specifies the trustee rights for the file or directory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path

## Remarks

*Sequence* should be set to 1 on the first request; for each subsequent request, it should be incremented by 1.

No matter how many extended trustee rights entries are returned, *TrusteeRights* are returned starting at offset 89 in the reply buffer.

## See Also

**Scan Directory for Trustees 0x2222 22 12 (page 458)**

# Scan File or Subdirectory for Trustees 0x2222 87 05

NetWare Servers: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (05)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	SearchAttribute	word (Lo-Hi)
12	SearchSequence	long (Lo-Hi)
16	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence	long (Lo-Hi)
12	ObjectIDCount	word (Lo-Hi)
14	TrusteeStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Set *SearchSequence* to zero on the first request. For each subsequent request, set *SearchSequence* to the returned *NextSearchSequence* value. When *NextSearchSequence* is -1, you have received all the trustees.



The server will send the maximum trustee structures (up to 20) until the last request. The last request contains a partial packet of trustee structures.

NWHandlePathStruct is of type [NetWareHandlePathStruct](#) (page 609).

# Scan Salvageable File List 0x2222 87 41

Generates a list of salvageable files from a path.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (41)	byte
8	NameSpace	byte
9	reserved	byte
10	ControlFlags	word (Lo-Hi)
12	ScanSequence	long (Lo-Hi)
16	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextScanSequence	long (Lo-Hi)
12	PurgeBase	long (Lo-Hi)
16	ScanItems	long (Lo-Hi)
20	ScanInfo	structure

## Parameters

### *ControlFlags*

(Request) Specifies whether to return the file's name:

0x00000001 ReturnFileName

### *NextScanSequence*

(Reply) Specifies the next ScanSequence number if more items are present in the subdirectory:

-1 No more items are present

### *PurgeBase*

(Reply) Specifies the NetWare directory base of the path.

### *ScanItems*

(Reply) Specifies the number of valid ScanInfo structures that are returned.

### *ScanInfo*

(Reply) Points to either [ScanInfoFileName \(page 622\)](#) or [ScanInfoFileNoFileName \(page 623\)](#), depending on the *ControlFlag* value.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

NWHandlePathStruct must point to a subdirectory path and is of type [NetWareHandlePathStruct \(page 609\)](#).

No file names or wildcards are allowed when you are searching for salvageable files or subdirectories.

Set *ScanSequence* to -1 on the first request. For each subsequent request, use the returned *SalvageableFileEntryNumber* field value from the [ScanInfoFileName \(page 622\)](#) structure or the [ScanInfoFileNoFileName \(page 623\)](#) structure.

If you are going to pass the returned list to **Purge Salvageable File List 0x2222 87 42**, you cannot set the ReturnFileName bit in *ControlFlag*. **Purge Salvageable File List** requires the list to contain back-to-back salvageable file entry numbers.

## See Also

[Purge Salvageable File \(old\) 0x2222 22 29 \(page 424\)](#), [Recover Salvageable File \(old\) 0x2222 22 28 \(page 436\)](#), [Scan Salvageable Files \(old\) 0x2222 22 27 \(page 474\)](#), [Purge Salvageable File List 0x2222 87 42 \(page 425\)](#)

# Scan Salvageable Files 0x2222 87 16

Scans a subdirectory for any files that have been deleted but not yet purged.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (16)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>ReturnInfoMask</b>	long (Lo-Hi)
14	ScanSequence	long (Lo-Hi)
18	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextScanSequence	long (Lo-Hi)
12	DeleteTime	word (Lo-Hi)
14	DeleteDate	word (Lo-Hi)
16	DeletorID	long (Hi-Lo)
20	ScanVolume	long (Lo-Hi)
24	ScanDirectoryBase	long (Lo-Hi)
28	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

NWHandlePathStruct must point to a subdirectory path and is of type [NetWareHandlePathStruct \(page 609\)](#).

No file names or wildcards are allowed when you are searching for salvageable files or subdirectories.

Set *ScanSequence* to -1 on the first request. For each subsequent request, use the returned *NextScanSequence* value.

# Scan Salvageable Files (old) 0x2222 22 27

Returns the salvageable file information for a file in the current directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (27)	byte
10	DirectoryHandle	byte
12	Sequence	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	Sequence	long (Lo-Hi)
12	Subdirectory	word (Lo-Hi)
14	Attributes	long (Lo-Hi)
18	UniqueID	byte
19	Flags	byte
20	NameSpace	byte
21	FileNameLen	byte
22	FileName	byte[12]
34	CreationDateAndTime	long (Lo-Hi)
38	OwnerID	long (Hi-Lo)
42	LastArchivedDateAndTime	long (Lo-Hi)
46	LastArchivedID	long (Hi-Lo)
50	LastUpdatedDateAndTime	long (Lo-Hi)
54	LastUpdatedID	long (Hi-Lo)

Offset	Content	Type
58	Reserved	byte[44]
106	InheritedRightsMask	word (Lo-Hi)
108	LastAccessedDate	word (Lo-Hi)
110	DeletedFileTime	long (Lo-Hi)
114	DeletedDateAndTime	long (Lo-Hi)
118	DeletedID	long (Hi-Lo)
122	Reserved	byte[16]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Server Does Not Support 386 Salvage Functions
255	0xFF	No More Salvageable Files In Directory

## Remarks

If *Sequence* is 0xFFFFFFFFh, **Scan Salvageable Files (old)** will start with a new search; otherwise, it will return the next file.

## See Also

**Purge Salvageable File (old) 0x2222 22 29 (page 424), Recover Salvageable File (old) 0x2222 22 28 (page 436)**

# Scan Volume's User Disk Restrictions 0x2222 22 32

Returns a list of the object restrictions for a specified volume.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (32)	byte
11	VolumeNumber	byte
12	Sequence	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfEntries	byte
(for each entry)		
9	ObjectID	long (Hi-Lo)
14	Restriction	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
152	0x98	Invalid Volume

## Remarks

All restrictions are in 4 KB blocks. A restriction can be zero. If a restriction is greater than 0x40000000, that entry has no restriction.

The maximum number of entries is 16. *Sequence* starts at zero and increments by the value of *NumberOfEntries*. If *NumberOfEntries* comes back as 16, there might be additional entries that



can be returned by reissuing the NCP with the sequent input parameter set to whatever it was issued with in the previous NCP request + 16.

## See Also

**Get Directory Disk Space Restriction 0x2222 22 35 (page 332), Set Directory Disk Space Restriction 0x2222 22 36 (page 487), Add User Disk Space Restriction 0x2222 22 33 (page 290), Remove User Disk Space Restrictions 0x2222 22 34 (page 438), Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366)**

# Search for a File 0x2222 64

Searches the specified server's directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (64)	byte
7	LastSearchIndex	word (Hi-Lo)
9	DirectoryHandle	byte
10	SearchAttributes	byte
11	FileNameLen	byte
12	FileName	byte[FileNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchIndex	word (Hi-Lo)
10	Reserved	word (Hi-Lo)
12	FileName	byte[14]
26	FileAttributes	byte
27	FileExecuteType	byte
28	FileLen	long (Hi-Lo)
32	CreationDate	word (Hi-Lo)
34	LastAccessDate	word (Hi-Lo)
36	LastUpdateDate	word (Hi-Lo)
38	LastUpdateTime	word (Hi-Lo)

# Return Values

Decimal	Hex	Description
0	0x00	Successful
137	0x89	No Search Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	No Files Found

# Remarks

You must have directory search privileges to the target directory or **Search for a File** will fail.

You can iteratively call **Search for a File**. On the first request, set *LastSearchIndex* to 0xFFFF. For subsequent requests, set *LastSearchIndex* to the *NextSearchIndex* returned by the previous request. You can then search for all files that match the specified search pattern. (Wildcard characters are supported.)

# See Also

**Scan File Information 0x2222 23 15 (page 462)**

# Search for File or Subdirectory 0x2222 87 03

Searches for a file or subdirectory, starting with the *SearchSequence* number returned by the **Initialize Search** request.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (03)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	SearchSequence[9]	byte
25	SearchPatternLen	byte
26	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	Reserved (0)	byte
18	NetWareInfoStruct	structure
xx	NetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*NetWareFileNameStruct* is always returned when you call **Search for File or Subdirectory**. With all other 87 requests, however, this field is not returned unless the Include File Name Structure bit is set in *ReturnInfoMask*.

You can iteratively call **Search for a File**. On the first request, set *LastSearchIndex* to 0xFFFF. For subsequent requests, set *LastSearchIndex* to the *NextSearchIndex* returned by the previous request. You can then search for all files that match the specified search pattern. (Wildcard characters are supported.)

## See Also

**Initialize Search 0x2222 87 02 (page 384)**

# Search for File or Subdirectory Set 0x2222 87 20

Searches for a file or subdirectory, starting with the *SearchSequence* number returned by **Initialize Search**.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (20)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	ReturnInfoCount	word (Lo-Hi)
18	SearchSequence[9]	byte
25	SearchPatternLen	byte
26	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	MoreEntriesFlag	byte
18	InfoCount	word (Lo-Hi)
20	<b>NetWareInformationStruct</b>	structure
xx	<b>NetWareFileNameStruct</b>	structure
yy	...	

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Search for File or Subdirectory Set** returns as many NetWareInfoStructs as are requested by *ReturnInfoCount* (or that fit in a packet). *InfoCount* is the count of NetWareInfoStructs that are returned.

*MoreEntriesFlag* is set to -1 when more entries are available; it is set to zero when there are no more entries.

To get the NetWareFileNameStruct, IncludeFileNameStruct must be set in *ReturnInfoMask*.

For example, **Search for File or Subdirectory Set** might return the following:

```
InfoCount = 2
IncludeFileNameStruct = TRUE

0   NetWareInfoStruct
+   NetWareFileNameStruct
+   NetWareInfoStruct
+   NetWareFileNameStruct
```

# Search for File or Subdirectory Set (Extended Errors) 0x2222 87 40

Searches for a file or subdirectory, starting with the *SearchSequence* number returned by **Initialize Search**.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (40)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	ReturnInfoCount	word (Lo-Hi)
18	SearchSequence[9]	byte
25	SearchPatternLen	byte
26	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	MoreEntriesFlag	byte
18	InfoCount	word (Lo-Hi)
20	<b>NetWareInformationStruct</b>	structure
xx	<b>NetWareFileNameStruct</b>	structure
yy	...	



# Return Values

Decimal	Hex	Description
0	0x00	Successful

# Remarks

**Search for File or Subdirectory Set (Extended Errors)** returns extended error codes. If the path ends with a subdirectory that doesn't exist, `ERR_INVALID_PATH` (0x899C) is returned, rather than 0x89FF. If a file exists but you have insufficient rights to see it, `ERR_ACCESS_DENIED` (0x89A8) is returned, rather than 0x89FF. To access these extended error codes, the operating system ORs 0x80000000 to the search attributes so that the extended error codes can be returned. You do not have to pass this bit.

**Search for File or Subdirectory Set (Extended Errors)** returns as many `NetWareInfoStructs` as are requested by *ReturnInfoCount* (or that fit in a packet). `InfoCount` is the count of `NetWareInfoStructs` that are returned.

*MoreEntriesFlag* is set to -1 when more entries are available; it is set to zero when there are no more entries.

To get the `NetWareFileNameStruct`, the `IncludeFileNameStruct` must be set in *ReturnInfoMask*.

For example, **Search for File or Subdirectory Set** might return the following:

```
InfoCount = 2
IncludeFileNameStruct = TRUE

0 NetWareInfoStruct
+ NetWareFileNameStruct
+ NetWareInfoStruct
+ NetWareFileNameStruct
```

# Set Compressed File Size 0x2222 90 12

Sets the size on a compressed file to the specified size.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (90)	byte
7	SubFuncStrucLen (11)	word (Hi-Lo)
9	SubFunction (12)	byte
10	NetWareFileHandle	byte[6]
16	SuggestedFileSize	long

## Reply Format

Offset	Content	Type
Reply header		
8	OldFileSize	long
12	NewFileSize	long

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Set Directory Disk Space Restriction 0x2222 22 36

Sets a disk restriction for the specified directory.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunction (36)	byte
10	DirHandle	byte
11	DiskSpaceLimit	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Space Limit
140	0x8C	No Set Privileges
191	0xBF	Invalid Name Space

## Remarks

If the restriction is zero, the restriction for the directory is cleared. If the restriction is a negative number, the disk space is set to zero. All restrictions are in 4 KB blocks.

## See Also

**Get Directory Disk Space Restriction 0x2222 22 35 (page 332), Add User Disk Space Restriction 0x2222 22 33 (page 290), Remove User Disk Space Restrictions 0x2222 22 34 (page 438), Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366), Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476)**

# Set Directory Entry Information 0x2222 22 37

Sets or changes the file or directory entry information to the values entered in *ChangeBits*.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
8	SubFuncStrucLen (148)	word (Hi-Lo)
9	SubFunctionCode (37)	byte
10	DirHandle	byte
11	SearchAttributes	byte
12	Sequence	long (Lo-Hi)
16	ChangeBits	long
20	Subdirectory	long (Lo-Hi)

## Request Format (DOS File Entry)

Offset	Content	Type
24	Attributes	long (Lo-Hi)
28	UniqueID	byte
29	Flags	byte
30	NameSpace	byte
31	NameLen	byte
32	Name	byte[12]
44	CreationDateAndTime	long (Lo-Hi)
48	OwnerID	long (Hi-Lo)
52	LastArchivedDateAndTime	long (Lo-Hi)
56	LastArchivedID	long (Hi-Lo)
60	LastUpdatedDateAndTime	long (Lo-Hi)
64	LastUpdatedID	long (Hi-Lo)

Offset	Content	Type
68	FileSize	long (Hi-Lo)
72	DataForkFirstFAT	long (Hi-Lo)
76	NextTrusteeEntry	long (Hi-Lo)
80	Reserved	byte[36]
116	InheritedRightsMask	word (Lo-Hi)
118	LastAccessedDate	word (Lo-Hi)
120	Reserved	byte[28]
140	PrimaryEntry	long (Lo-Hi)
144	NameList	long (Lo-Hi)

## Request Format (DOS Subdirectory Entry)

Offset	Content	Type
24	Attributes	long (Lo-Hi)
28	UniqueID	byte
29	Flags	byte
30	NameSpace	byte
31	DirectoryNameLen	byte
32	DirectoryName	byte[12]
44	CreationDateAndTime	long (Lo-Hi)
48	OwnerID	long (Hi-Lo)
52	LastArchivedDateAndTime	long (Lo-Hi)
56	LastArchivedID	long (Hi-Lo)
60	LastModifiedDateAndTime	long (Lo-Hi)
64	NextTrusteeEntry	long (Hi-Lo)
68	Reserved	byte[48]
116	MaximumSpace	long (Lo-Hi)
120	InheritedRightsMask	word (Lo-Hi)
122	Undefined	byte[26]

## Request Format (Macintosh Name Space Entry)

Offset	Content	Type
24	MACFileAttributes	long
28	MACUniqueID	long
29	MACFlags	byte
30	MACMyNameSpace	byte
31	MACFileNameLen	byte
32	MACFileName	byte[32]
64	ResourceFork	long
68	ResourceForkSize	long
72	FinderInfo	byte[32]
104	ProDosInfo	byte[6]
110	Reserved	byte[38]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Parameter
140	0x8C	No Set Privileges
191	0xBF	Invalid Name Space

# Set Directory Handle 0x2222 22 00

Changes the directory pointed to by *TargetDirectoryHandle* to the directory pointed to by *SourceDirectoryHandle*.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (4 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (0)	byte
10	TargetDirectoryHandle	byte
11	TargetDirectoryHandle	byte
12	DirectoryPathLen	byte
13	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
250	0xFA	Temporary Remap Error
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

The target directory handle must already exist.

To create a new directory handle, you must call 0x2222 22 18, 0x2222 22 19, or 0x2222 22 22.

You can specify the complete directory path and pass zero to *SourceDirectoryHandle*.

The directory handle returned by the server will be the same handle as *TargetDirectoryHandle*.

*AccessRightsMask* contains the client's effective access rights for the specified directory.

## See Also

**[Alloc Permanent Directory Handle 0x2222 22 18 \(page 291\)](#), [Alloc Temporary Directory Handle 0x2222 22 19 \(page 295\)](#), [Deallocate Directory Handle 0x2222 22 20 \(page 310\)](#), [Alloc Special Temporary Directory Handle 0x2222 22 22 \(page 293\)](#)**



# Set Directory Information 0x2222 22 25

Modifies information about a directory.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (22)	byte
7	SubFuncStrucLen (12 + DirectoryPathLen)	word (Hi-Lo)
9	SubFunctionCode (25)	byte
10	DirectoryHandle	byte
11	CreationDate	word (Lo-Hi)
13	CreationTime	word (Lo-Hi)
15	OwnerID	long (Hi-Lo)
19	MaximumAccessRightsMask	byte
20	DirectoryPathLen	byte
21	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
255	0xFF	Failure

## Remarks

You can call **Set Directory Information** to restore a directory that has been destroyed and is being regenerated from some backup medium.

*CreationDate*, *CreationTime*, and *MaximumAccessRightsMask* can be set by any client that has access control and modify rights in the directory's parent directory.

OwnerID can be changed only by a client that is a supervisor of the object.

## See Also

**Get Directory Information 0x2222 22 45 (page 338)**

# Set File Attributes 0x2222 70

Modifies a file’s attributes.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (70)	byte
7	NewFileAttributes	byte
8	DirectoryHandle	byte
9	SearchAttributes	byte
10	FileNameLen	byte
11	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

You must have file modification privileges in the target directory.  
The target file must not be in used by other clients.

Once set, the execute only attribute on a file cannot be reset.

**Set File Attributes** supports wildcard attributes.

For NetWare 3.0, **Set File Attributes** can also be called to modify directory attributes.

## See Also

**Scan File Information 0x2222 23 15 (page 462)**

# Set File Extended Attributes 0x2222 79

Modifies a file’s attributes and sets the NetWare extended attributes byte.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (79)	byte
7	NewFileAttributes	byte
8	DirectoryHandle	byte
9	SearchAttributes	byte
10	FileNameLen	byte
11	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
140	0x8C	No Set Privileges
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
255	0xFF	Failure, No Files Found

## Remarks

The bits of the extended attribute byte are as follows:

Bit	Description
0-3	If set, Search Mode (which is comprised of the first three bits), is valid only with NetWare 2.0a and above.
4	If set, Transaction prompts NetWare's TTS to track all writes to the file during a transaction. A transactional file cannot be deleted or renamed.
5	If set, Index prompts NetWare to index the file's File Allocation Tables (FATs), which reduces the time it takes to access the file.
6	Can be set only by a user with security equivalence to supervisor.
7	Can be set only by a user with security equivalence to supervisor.

## See Also

**Scan File Information 0x2222 23 15 (page 462), Set File Attributes 0x2222 70 (page 495)**

# Set File Information 0x2222 23 16

Sets a file’s status information.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
8	SubFuncStrucLen (82 + FileNameLen)	word (Hi-Lo)
9	SubFunctionCode (16)	byte
10	FileAttributes	byte
11	ExtendedFileAttributes	byte
12	FileSize	long (Hi-Lo)
16	FileCreationDate	word (Hi-Lo)
18	LastAccessedDate	word (Hi-Lo)
20	LastUpdateDate	word (Hi-Lo)
22	LastUpdateTime	word (Hi-Lo)
24	UniqueOwnerID	long (Hi-Lo)
28	LastArchiveDate	word (Hi-Lo)
30	LastArchiveTime	word (Hi-Lo)
32	Reserved	byte[56]
88	DirectoryHandle	byte
89	SearchAttributes	byte
90	FileNameLen	byte
91	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle

Decimal	Hex	Description
140	0x8C	No Set Privileges
142	0x8E	All Files In Use
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
252	0xFC	No Such Object
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	No Files Found, Failure

## Remarks

**Set File Information** is primarily used by archive programs that need to restore a file's complete profile to an earlier state.

Any client with file modification privileges in the target directory can set a file's attributes, execute type, file creation date, last access date, last update date and time, and last archive date and time.

*FileSize* is always ignored.

A client that is an object supervisor of the file can set a file's owner ID, the last archive date and time, and the 56 bytes of undefined information.

## See Also

**Set File Extended Attributes 0x2222 79 (page 497), Set File Attributes 0x2222 70 (page 495), Scan File Information 0x2222 23 15 (page 462), Set File Time Date Stamp 0x2222 75 (page 501)**



# Set File Time Date Stamp 0x2222 75

Changes the time date stamp on the specified file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (75)	byte
7	Reserved	byte
8	FileHandle	byte[6] (Hi-Lo)
14	ForgedTime	word (Hi-Lo)
16	ForgedDate	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
150	0x96	Server Out Of Memory

## Remarks

*FileHandle* requires the left most or most significant 4 bytes in Hi-Lo order.

# Set Huge NS Information 0x2222 87 27

Sets huge name space information.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (27)	byte
8	NameSpace	byte
9	VolumeNumber	byte
10	DirectoryBase	long (Lo-Hi)
14	HugeMask	long (Lo-Hi)
18	HugeStateInfo	byte[16]
34	HugeDataLen	long (Lo-Hi)
38	HugeData[]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NextHugeStateInfo	byte[16]
24	HugeDataUsed	long (Lo-Hi)

## Parameters

### *HugeStateInfo*

Is used only by the NFS name space.

### *HugeDataLen*

Specifies the length of the data that will be returned in the reply buffer.

### *HugeDataUsed*

Specifies the number of bytes consumed by the name space out of the total data bytes that were sent to the name space.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Set Huge NS Information** is called only when the name space has indicated that there is huge information available (by calling **Query NS Information Format 0x2222 87 23**).

## See Also

**Get Huge NS Information 0x2222 87 26 (page 353), Query NS Information Format 0x2222 87 23 (page 427)**

# Set NS Information 0x2222 87 25

Sets the specified name space information.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (25)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	VolumeNumber	byte
11	DirectoryBase	long (Lo-Hi)
15	NSInfoBitMask	long (Lo-Hi)
19	NSSpecificInfo[512]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
139	0x8B	No Rename Privileges

## Remarks

**Set NS Information** is passed to the name space NLM and is an expensive time user on the server.

The information that is set depends on *NSInfoBitMask*.

## See Also

**Get NS Information 0x2222 87 19 (page 357), Query NS Information Format 0x2222 87 23 (page 427)**

# Set Short Directory Handle 0x2222 87 09

Sets a short directory handle.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunction (09)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	DstDirHandle	byte
11	Flags	byte
12	NWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Flags Value		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	<b>EnhNetWareFileNameStruct</b>	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If the Flags field is set to 0x40, the reply contains the four items listed in the Reply Format section.  
If the input Flags field is zero, no data is returned.

NWHandlePathStruct is of type **NetWareHandlePathStruct** (page 609).

## See Also

[Allocate Short Directory Handle 0x2222 87 12 \(page 297\)](#)

## Write to a File 0x2222 87 65

Writes a block of data to a file.

**NetWare Servers:** 6.x

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (65)	byte
8	FileHandle	long (Hi-Lo)
12	StartingByteOffset	UINT64 (Hi-Lo)
20	BytesToWrite	word (Hi-Lo)
22	Data	byte[BytesToWrite]

### Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
148	0x94	No Write Privileges
149	0x95	File Detached
162	0xA2	I/O Lock Error
253	0xFD	Bad Station Number
255	0xFF	I/O Bound Error

## Remarks

Note that *StartingByteOffset* in the request can be 64 bits. Requests made to NSS volumes support writing at true 64-bit offsets. Requests made to traditional volumes support writing at 32-bit offsets. Writing at offsets greater than 32 bits on a traditional volume result in an error.

If you do not have write access to the specified file or if some portion of the targeted byte block is locked for use by a different client, **Write to a File** will fail.

You are constrained by the current negotiated file buffer size (see *Negotiate Buffer Size 0x2222 33*). You cannot write more bytes of data than will fit in the buffer size. Also, you cannot write a data block that straddles a buffer size 4 KB boundary in the file.

For example, if the current buffer size were 4,096 bytes and you want to write 4,200 bytes, starting at file offset 4000, you must issue three write requests. The first request would write 96 bytes starting at offset 4000. The second request would write 4,096 bytes starting at offset 4,096. The last request would write 8 bytes starting at offset 8,192.

If you do not write any bytes at position zero, the files will be truncated.

*FileHandle*, *StartingByteOffset*, and *BytesToWrite* require the input to be in Hi-Lo order.

## See Also

**Negotiate Buffer Size 0x2222 33 (page 213)**, **Read File 0x2222 87 64 (page 429)**



# Write to a File 0x2222 73

Writes a block of data to a file.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (73)	byte
7	Reserved	byte
8	FileHandle	byte[6] (Hi-Lo)
14	StartingByteOffset	long (Hi-Lo)
18	BytesToWrite	word (Hi-Lo)
20	Data	byte[BytesToWrite]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
131	0x83	Hard I/O Error
136	0x88	Invalid File Handle
148	0x94	No Write Privileges
149	0x95	File Detached
162	0xA2	I/O Lock Error
255	0xFF	I/O Bound Error

## Remarks

If you do not have write access to the specified file or if some portion of the targeted byte block is locked for use by a different client, **Write to a File** will fail.

You are constrained by the current negotiated file buffer size (see Negotiate Buffer Size 0x2222 33). You cannot write more bytes of data than will fit in the buffer size. Also, you cannot write a data block that straddles a buffer size 4 KB boundary in the file.

For example, if the current buffer size were 4,096 bytes and you want to write 4,200 bytes, starting at file offset 4000, you must issue three write requests. The first request would write 96 bytes

starting at offset 4000. The second request would write 4,096 bytes starting at offset 4,096. The last request would write 8 bytes starting at offset 8,192.

If you don't write any bytes at position 0, the files will be truncated.

## See Also

**Negotiate Buffer Size 0x2222 33 (page 213)**

# 22

## Enhanced NCPs

Enhanced File System NCPs enable you to send and return path string data in either UTF8 or ASCII format.

This section describes each of the Enhanced File System NCPs, their Request and Reply formats, and Return Values.

# Add Trustee Set to File or Subdirectory 0x2222 89 10

Adds a trustee to a file or subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (10)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	TrusteeRightsMask	word (Lo-Hi)
14	ObjectIDCount	word (Lo-Hi)
16	<b>TrusteeStruct</b>	structure
xx	EnhNWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If TrusteeRightsMask is set to -1, the individual TrusteeRights should be set to the appropriate rights for each entry in the TrusteeStruct.

EnhNWHandlePathStruct is of type **EnhNetWareHandlePathStruct** (page 592) and begins directly after the last entry in the TrusteeStruct. You can figure out the amount of space the TrusteeStruct occupies by multiplying the number in the ObjectIDCount field by the sizeof TrusteeStruct (6 bytes).

Ensure that the number of input TrusteeStruct entires balances with the length of the path components in the EnhNWHandlePathStruct. If the path components don't occupy much space, more TrusteeStruct entries can be passed back. If the length of the path components take up a large amount of buffer space, reduce the number of TrusteeStruct entries.

# Allocate Short Directory Handle 0x2222 89 12

Allocates a short directory handle.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (12)	byte
8	NameSpace	byte
9	DstNameSpace	byte
10	AllocateMode	word (Lo-Hi)
12	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	DirectoryHandle	byte
9	VolumeNumber	byte
10	reserved[4] (0)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

DstNameSpace is ignored unless the high bit (0x8000) is set. If the high bit is not set, the NameSpace field specifies what name space the allocation takes place in. DstNameSpace allows you to call the NCP in one name space and allocate a directory handle in another name space.

AllocateMode can have the following values:

Bit Number	Value	Type of Handle
0	0x00	Permanent Handle
1	0x01	Temporary Handle
2	0x02	Special Temporary Handle
3-13		reserved
14	0x4000	Activates Reply Level 2
15	0x8000	Activates DstNameSpace input parameter

If 0x4000 is set, the following reply format is used:

Offset	Content	Type
Reply header		
8	Volume	long
12	DirectoryBase	long
16	DOSDirectoryBase	long
20	NameSpace	long
24	DirectoryHandle	byte

## See Also

**Set Short Directory Handle 0x2222 89 09 (page 567)**

# Delete a File or Subdirectory 0x2222 89 08

Deletes a file or subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (80)	byte
7	SubFunction (08)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	EnhNWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type **EnhNetWareHandlePathStruct** (page 592).

# Delete Trustee Set from File or SubDirectory 0x2222 89 11

Deletes a trustee set from a file or from a subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (11)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	ObjectIDCount	word (Lo-Hi)
12	TrusteeStruct	structure
xx	EnhNWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#) and begins directly after the last entry in the TrusteeStruct. You can figure out the amount of space the TrusteeStruct occupies by multiplying the number in the ObjectIDCount field by the sizeof TrusteeStruct (6 bytes).

Ensure that the number of input TrusteeStruct entires balances with the length of the path components in the EnhNWHandlePathStruct. If the path components don't occupy much space, more TrusteeStruct entries can be passed back. If the length of the path components take up a large amount of buffer space, reduce the number of TrusteeStruct entries.



# Enhanced Enumerate Extended Attribute 0x2222 89 54

Enumerates extended attributes.

NetWare Servers: 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (54)	byte
8	Flags	word (Lo-Hi)
10	EAHandleStruct	structure
18	InspectSize	long (Lo-Hi)
22	EnumerateSequence	word (Lo-Hi)
24	DataTypeFlag	byte
25	KeyLength	word (Lo-Hi)
27	Key[ ]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	reserved (0)	word (Lo-Hi)
30	reserved (0)	word (Lo-Hi)

## Information Level=1

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	NextEnumerateSequence	word (Lo-Hi)
30	EnumEAStructCount	word (Lo-Hi)
32	EnumEAStruct_Lvl1[ ]	structure

## Information Level=6

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)
24	NewEAHandle	long (Lo-Hi)
28	reserved (0)	word (Lo-Hi)
30	reserved (0)	word (Lo-Hi)
32	EnumEAStruct_Lvl6[ ]	structure

## Information Level=7

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlEAs	long (Lo-Hi)
16	TtlEAsDataSize	long (Lo-Hi)
20	TtlEAsKeySize	long (Lo-Hi)

Offset	Content	Type
24	NewEAHandle	long (Lo-Hi)
28	NextEnumerateSequence	word (Lo-Hi)
30	EnumEAStructCount	word (Lo-Hi)
32	EnumEAStruct_Lvl7[ ]	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful
201	0xC9	ERR_EA_NOT_FOUND
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
		ERR_INTERNAL_FAILURE
		ERR_NO_ALLOC_SPACE
		ERR_UNKNOWN_REQUEST

## Remarks

When using a level 6 enumerate request, the fields TtlEAsDataSize and TtlEAsKeySize contain the size of the value and key specified only and TtlEAs contains a 1. The KeyPages and ValuePages contain the number of pages of extended directory space that was used.

DataTypeFlag indicates what format the key name is in:

- 0 ASCII
- 1 UTF8

KeyLength is always a WORD (two bytes), regardless of whether DataTypeFlag is 0 or 1.

If KeyLength equals 0, a list of all EAs (including information about each EA) is passed back. If a key is used, only information level 6 can be used.

On the first call, EnumerateSequence must be set to 0. On subsequent calls, pass the NextEnumerateSequence from the reply to EnumerateSequence.

# Enhanced Get Object Effective Rights 0x2222 89 50

Returns an object's effective access rights to a specified directory or file.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (50)	byte
8	NameSpace	byte
9	ObjectID	long (Hi-Lo)
13	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
9	MyEffectiveRights	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The client must be either a supervisor, a manager of the object for which rights are requested, or the object itself.

**Enhanced Get Object Effective Rights** is based on **Get Object Effective Rights for Directory Entry 0x2222 22 50**.

# Enhanced Read Extended Attribute 0x2222 89 53

Reads extended attributes.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (53)	byte
8	Flags	word (Lo-Hi)
10	ECHandleStruct	structure
18	ReadPosition	long (Lo-Hi)
22	InspectSize	long (Lo-Hi)
26	DataTypeFlag	byte
27	MaxReadDataReplySize	word (Lo-Hi)
29	KeyLength	word (Lo-Hi)
35	Key[ ]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	TtlValuesLength	long (Lo-Hi)
16	NewEAHandle	long (Lo-Hi)
20	AccessFlag	long (Lo-Hi)
24	ValueLength	word (Lo-Hi)
26	Vaue[ ]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
201	0xC9	ERR_EA_NOT_FOUND
206	0xCE	ERR_EA_BAD_DIR_NUM
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED
		ERR_EA_INTERNAL_FAILURE
		ERR_EA_INSPECT_FAILURE
		ERR_NO_ALLOC_SPACE
		ERR_INVALID_FILE_HANDLE
		ERR_NO_SET_PRIVILEGES
		ERR_INVALID_PATH

## Remarks

ReadData is always returned in 128-byte chunks.

DataTypeFlag indicates what format the key name is in:

- 0 ASCII
- 1 UTF8

KeyLength is always a WORD (two bytes), regardless of whether DataTypeFlag is 0 or 1.

MaxReadDataReplySize allows you to specify how big the reply buffer is. If the connection is an IP connection, the maximum is 1408 bytes. If the connection is IPX, the maximum is 512 bytes (or four 128-byte chunks). You need to explicitly indicate how big your reply buffer is. If your reply buffer is bigger than what the NCP supports, the reply size is set to the largest size that the NCP supports. Currently, the largest size is 1408 bytes + 18 bytes (the rest of the reply packet) = 1426 bytes.

The key is sent only on the first read. Subsequent reads should not contain the key. Set KeyLength to zero after the first read.

# Enhanced Scan Volume Trustee Object Paths 0x2222 89 71

Returns a path on the specified volume where the specified object is a trustee.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (71)	byte
8	Volume	long (Lo-Hi)
12	ObjectID	long (Hi-Lo)
16	SequenceNumber	word (Lo-Hi)
18	DataTypeFlag	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NextSequenceNumber	word (Lo-Hi)
10	ObjectID	long (Hi-Lo)
14	TrusteeAccessMask	byte
15	DirectoryPathLen	byte (ASCII) or WORD (Lo-Hi for UTF8)
16/17	DirectoryPath	byte[DirectoryPathLen]

## Return Values

Decimal	Value	Description
0	0x00	Successful
147	0x93	No Read Privileges
150	0x96	Server Out Of Memory
161	0xA1	Directory I/O Error
240	0xF0	Illegal Wildcard

Decimal	Value	Description
241	0xF1	Bindery Security
242	0xF2	No Object Read
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Hard Failure

## Remarks

The returned path is also known as the trustee path. You can call **Enhanced Scan Volume Trustee Object Paths** to determine all paths on a specified volume where the object ID number appears in a directory's trustee list.

DataTypeFlag indicates what format the key name is in:

- 0 ASCII
- 1 UTF8

By default, the length-preceding field is a byte. If data type is UTF8 format, the length-preceding field is a WORD (two bytes) in Lo-Hi order

Set the client's SequenceNumber to -1L (word -1) initially. In subsequent requests, set SequenceNumber to the value returned in NextSequenceNumber of the previous reply.

ObjectID must contain the number of the object (user) whose trustee directory paths should be returned.

The file server returns the directory path in the following format:

volume:directory/subdirectory/subdirectory/. . .

Directory paths are not returned in any special order. If all directory paths where the object is listed as a trustee have been returned, the server returns a directory path length of 0.

A client who is a supervisor (or equivalent) can make this request to investigate the trustee directory rights of any object in the bindery. Any client can make this request to determine its own trustee paths, or the trustee paths of any object to which the client is security equivalent.



# Enhanced Set NS Information 0x2222 89 25

Sets the specified name space information.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (25)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	VolumeNumber	byte
11	DirectoryBase	long (Lo-Hi)
15	NSInfoBitMask	long (Lo-Hi)
19	DataTypeFlag	byte
20	NSSpecificInfo[512]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
139	0x898B	No Rename Privileges

## Remarks

**Enhanced Set NS Information** is passed to the name space NLM. It is an expensive time user on the server.

DataTypeFlag indicates what format the file name in the NewFileName field is in:

- 0 ASCII
- 1 UTF8

If the data format is UTF8 and the mask indicates that the name is returned, the length-preceding field is a WORD (two bytes) in Lo-Hi order. Otherwise, the length-preceding field is a byte.

NSSpecificInfo is defined as:

```
BYTE   FileName&LengthByte[13]:  
LONG   FileAttributes; (Lo-Hi)
```

WORD	CreateDate; (Lo-Hi)
WORD	CreateTime; (Lo-Hi)
LONG	OwnerID; (Hi-Lo)
WORD	ArchiveDate; (Lo-Hi)
WORD	ArchiveTime; (Lo-Hi)
LONG	ArchiveID; (Hi-Lo)
WORD	ModifyDate; (Lo-Hi)
WORD	ModifyTime; (Lo-Hi)
LONG	ModifyID; (Hi-Lo)
WORD	LastAccessDate; (Lo-Hi)
LONG	InheritedRightsMask; (Lo-Hi)
LONG	MaximumSpace; (Lo-Hi)

## Parameters

srcNameSpace

(Request) Specifies the source name space used to parse path information.

dstNameSpace

(Request) Specifies the name space for the returned directory base and volume number information if dstNSIndicator is set to Jn; otherwise, this parameter is ignored.

If dstNSIndicator is set to Jn, **Generate Directory Base and Volume Number** returns a directory base for the name space specified by dstNameSpace.

dstNSIndicator

(Request) Specifies to return the directory base and volume number in the name space specified by dstNameSpace if this parameter is set to Jn; otherwise, the directory base and volume number information are returned in the name space specified by srcNameSpace.

# Enhanced Write Extended Attribute 0x2222 89 52

Writes extended attributes.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (52)	byte
8	Flags	word (Lo-Hi)
10	EAHandleStruct	structure
18	TtlWriteDataSize	long (Lo-Hi)
22	WritePosition	long (Lo-Hi)
26	AccessFlag	long (Lo-Hi)
30	DataTypeFlag	byte
31	ValueLength	word (Lo-Hi)
33	KeyLength	word (Lo-Hi)
35	Key[ ]	byte
xx	Value[ ]	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ErrorCode	long (Lo-Hi)
12	BytesWritten	long (Lo-Hi)
16	NewEAHandle	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
200	0xC8	ERR_MISSING_EA_KEY
201	0xC9	ERR_EA_NOT_FOUND
203	0xCB	ERR_EA_NO_KEY_NO_DATA
206	0xCE	ERR_EA_BAD_DIR_NUM
207	0xCF	ERR_INVALID_EA_HANDLE
209	0xD1	ERR_EA_ACCESS_DENIED
210	0xD2	ERR_DATA_PAGE_ODD_SIZE
211	0xD3	ERR_EA_VOLUME_NOT_MOUNTED
212	0xD4	ERR_BAD_PAGE_BOUNDARY
		ERR_EA_INTERNAL_FAILURE
		ERR_EA_WRITE_OUT_OF_RANGE
		ERR_NO_SET_PRIVILEGES
		ERR_INSUFFICIENT_SPACE
		ERR_HARD_FAILURE
		ERR_INVALID_PATH

## Remarks

The key is sent only on the first write. Subsequent writes should not contain the key. Set KeyLength to zero after the first write.

DataTypeFlag indicates what format the key name is in:

- 0 ASCII
- 1 UTF8

KeyLength is always a WORD (two bytes), regardless of whether DataTypeFlag is 0 or 1.

# Generate Directory Base and Volume Number 0x2222 89 22

Generates a directory base and volume number.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (22)	byte
8	srcNameSpace	byte
9	dstNameSpace	byte
10	dstNSIndicator	word
12	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NSDirectoryBase	long (Lo-Hi)
12	DOSDirectoryBase	long (Lo-Hi)
16	VolumeNumber	byte

## Parameters

srcNameSpace

(Request) Specifies the source name space used to parse path information.

dstNameSpace

(Request) Specifies the name space for the returned directory base and volume number information if dstNSIndicator is set to Jn; otherwise, this parameter is ignored.

If dstNSIndicator is set to Jn, **Generate Directory Base and Volume Number** returns a directory base for the name space specified by dstNameSpace.

dstNSIndicator

(Request) Specifies to return the directory base and volume number in the name space specified by dstNameSpace if this parameter is set to Jn; otherwise, the directory base and volume number information are returned in the name space specified by srcNameSpace.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct](#) (page 592).

# Get Directory Disk Space Restriction 0x2222 89 39

Scans for the amount of disk space assigned to all directories between the current directory and the root directory and returns information about the restrictions along the directory path.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (39)	byte
8	NameSpace	byte
9	reserved	byte[2]
11	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NumberOfEntries	byte (for each entry)
9	DirDiskSpaceResList	structure

## Parameters

NumberOfEntries

(Reply) Specifies the number of DirDiskSpaceResList structures that follow.

Max

(Reply) Specifies the maximum amount of space assigned to a directory.

Current

(Reply) Specifies the amount of space assigned to the directory minus the amount of space used by the directory and its subdirectories.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

To find the actual amount of space available for a directory, scan all the current entries and use the smallest entry. Directories that have no restrictions do not return any information. If no entries are returned, no space restrictions exist for the specified directory. All restrictions are in 4 KB blocks.

When Max is 0x7FFFFFFF, there is no restriction on an entry; however, you can still calculate the space in use by subtracting Current from Max.

When Max is negative, the limit is zero. When Current is negative, the amount of space assigned to the directory is really zero. These two fields are allowed to be negative so you can still generate a valid IN USE value.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

The target must be a subdirectory or an error is returned.

There is one DirDiskSpaceRestList for each entry back to the root from the current entry.



# Get Effective Directory Rights 0x2222 89 29

Returns the access rights the client has for the specified directory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (29)	byte
8	NameSpace	byte
9	DestNameSpace	byte
10	SearchAttributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	MyEffectiveRights	word (Lo-Hi)
10	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct](#) (page 592).

# Get Full Path String 0x2222 89 28

Returns the full path.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (28)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	PathCookie	structure
20	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	PathCookie	structure
18	PathComponentSize	word (Lo-Hi)
20	PathComponentCount	word (Lo-Hi)
22	PathComponent	structure

## Parameters

- PathCookie
- (Reply) Specifies the sequence path components that are too long to fit in a reply packet.
- PathComponentSize
- (Reply) Specifies the total byte size of PathComponent.
- PathComponentCount
- (Reply) Specifies the number of components in the reply packet.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The path returned by **Get Full Path String** is returned in reverse order, with the root being the last component and the current directory being the first component.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

The length field of the PathComponent structure is determined by the DataTypeFlag that is passed in to the NCP request. If the data type is ASCII format, the length field is 1 byte. If the data type is UTF8, the length field is two bytes (WORD).

# Get NS Information 0x2222 89 19

Returns information for the specified name space.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (19)	byte
8	SrcNameSpace	byte
9	DstNameSpace	byte
10	DataTypeFlag	byte
11	VolumeNumber	byte
12	DirectoryBase	long (Lo-Hi)
16	NSInfoBitMask	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NSSpecificInfo[512]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Get NS Information** is passed to the name space NLM. It is an expensive time user on the server.

DataTypeFlag indicates what format the file name in the NewFileName field is in:

- 0 ASCII
- 1 UTF8

If the data format is UTF8 and the mask indicates that the name is returned, the length-preceding field is a WORD (two bytes) in Lo-Hi order. Otherwise, the length-preceding field is a byte.

NSSpecificInfo is defined as:

```
BYTE    FileName&LengthByte[13]:
LONG    FileAttributes; (Lo-Hi)
WORD    CreateDate; (Lo-Hi)
WORD    CreateTime; (Lo-Hi)
LONG    OwnerID; (Hi-Lo)
WORD    ArchiveDate; (Lo-Hi)
WORD    ArchiveTime; (Lo-Hi)
LONG    ArchiveID; (Hi-Lo)
WORD    ModifyDate; (Lo-Hi)
WORD    ModifyTime; (Lo-Hi)
LONG    ModifyID; (Hi-Lo)
WORD    LastAccessDate; (Lo-Hi)
LONG    InheritedRightsMask; (Lo-Hi)
LONG    MaximumSpace; (Lo-Hi)
```

# Initialize Search 0x2222 89 02

Initializes the search for a file or subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (02)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte
9	DirectoryNumber	long (Lo-Hi)
13	EntryNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

This search request is a stateless search.

SearchSequenceDefinition is a 9-byte field, which contains a 1-byte volume number, a 4-byte directory number, and a 4-byte current directory number. SearchSequence is generated by the server only on each completed search request.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

# Modify DOS Attributes on a File or Subdirectory 0x2222 89 35

Sets the DOS attributes field on an entry by using an apply mask.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunctionCode (35)	byte
8	NameSpace	byte
9	Flags	byte
10	SearchAttributes	word (Lo-Hi)
12	AttributeMask	long (Lo-Hi)
16	Attributes	long (Lo-Hi)
20	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	ItemsChecked	long (Lo-Hi)
12	ItemsChanged	long (Lo-Hi)
16	AttributeValidFlag	long (Lo-Hi)
20	NewAttributes	long (Lo-Hi)

## Parameters

- ItemsChecked  
(Reply) Specifies the number of items that were scanned according to the input criteria.
- ItemsChanged  
(Reply) Specifies the number of items whose attributes were changed.
- AttributeValidFlag  
(Reply) Specifies whether NewAttributes is valid.

NewAttributes

(Reply) Specifies the new value of the entry's attributes.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

A client doesn't need to get the old attributes before calling **Modify DOS Attributes on a File or Subdirectory**.

**Modify DOS Attributes on a File or Subdirectory** is similar to **Modify File or Subdirectory DOS Information** 0x2222 89 07 but has the additional capability to modify files with a wildcard.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

If AllowWildCardsBit (0x00000001) is set in Flags, wildcards can be used in EnhNWHandlePathStruct. If wildcards are permitted, only the first matching entry is returned in the reply packet.

Even if AttributeValidFlag is not valid, the entry's attributes are still changed.



# Modify File or Subdirectory DOS Information 0x2222 89 07

Modifies DOS information while the client is in another name space.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (07)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	SearchAttributes	word (Lo-Hi)
12	ModifyDOSInfoMask	long (Lo-Hi)
16	ModifyDOSInfoStruct	structure
54	EnhNWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Modify File or Subdirectory** does not change the name of a file or subdirectory. Passing 0x1 to ModifyDOSInfoMask has no effect.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

# Obtain File or Subdirectory Information 0x2222 89 06

Returns information for a file or subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (06)	byte
8	NameSpace	byte
9	DestNameSpace	byte
10	SearchAttributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

The data values in NetWareInfoStruct, from the least significant byte to the most significant byte, are:

- ♦ First 5 bits indicate the day, from 1-31.
- ♦ Next 4 bits indicate the month, from 1-12.

- ♦ Last 7 bits indicate the year, with 0=1980 and 20=2000.

The time values in NetWareInfoStruct, from the least significant byte to the most significant byte, are:

- ♦ First 5 bits indicate the number of 2-second intervals, from 0-29, so that 59 and 60 seconds are both indicated by 29.
- ♦ Next 6 bits indicate the minute, from 0-59.
- ♦ Last 5 bits indicate the hour, from 0-23.

SearchAttributes is currently not used. Regardless of what value is passed in to this field, information on the file or subdirectory is returned if the file or subdirectory exists.

# Open/Create File or Subdirectory 0x2222 89 01

Creates or opens the specified file, depending on the OpenCreateMode field. Subdirectories can be created (but not opened) by the client.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (01)	byte
8	<b>Namespace</b>	byte
9	<b>OpenCreateMode</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	CreateAttributes	long (Lo-Hi)
20	<b>DesiredAccessRights</b>	word (Lo-Hi)
22	EnhNWHandlePathStruct	structure

EnhNWHandlePathStruct is of type **EnhNetWareHandlePathStruct** (page 592).

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	long (Hi-Lo)
12	OpenCreateAction	byte
13	Reserved	byte
14	NetWareInfoStruct	structure
xx	<b>EnhNetWareFileNameStruct</b>	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Open/Create File or Subdirectory 0x2222 89 30

Creates or opens the file (depending on OpenCreateMode).

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (30)	byte
8	<b>Namespace</b>	byte
9	<b>DataStream</b>	byte
10	<b>OpenCreateMode</b>	byte
11	Reserved	byte
12	<b>SearchAttributes</b>	word (Lo-Hi)
14	Reserved	word (Lo-Hi)
16	<b>ReturnInfoMask</b>	long (Lo-Hi)
20	CreateAttributes	long (Lo-Hi)
24	<b>DesiredAccessRights</b>	word (Lo-Hi)
26	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	<b>OpenCreateAction</b>	byte
13	Reserved	byte
14	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

# Parameters

- NameSpace**  
(Request) Specifies the name space (see “NameSpace Values” on page 636).
- DataStream**  
(Request) Specifies the data stream number if the name space is Macintosh (see “DataStream Values” on page 633).
- OpenCreateMode**  
(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “OpenCreateMode Values” on page 636).
- SearchAttributes**  
(Request) Specifies the create and open options (see “SearchAttributes Values” on page 639).
- ReturnInfoMask**  
(Request) Specifies the information you want returned about a file or subdirectory.
- CreateAttributes**  
(Request) Specifies the DOS name space attributes.
- DesiredAccessRights**  
(Request) Specifies the access rights of the file that is being created (see “DesiredAccessRights Values” on page 633).
- FileHandle**  
(Reply) Specifies the file handle of the file being opened or created.
- OpenCreateAction**  
(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “OpenCreateAction Values” on page 636).
- NetWareInfoStruct**  
(Reply) Points to the NetWareInformationStructure.
- EnhNetWareFileNameStruct**  
(Reply) Points to the EnhNetWareFileNameStruct (page 591), which contains the name and length for the file or subdirectory.

# Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory** replaces **Open/Create File or Subdirectory** 0x2222 89 01. Subdirectories can be created, but not opened, by the client.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct](#) (page 592).



# Open/Create File or Subdirectory with Callback 0x2222 89 32

Creates or opens the file and sets a callback flag.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (32)	byte
8	Namespace	byte
9	OpenCreateMode	byte
10	Search Attributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	CreateAttributes	long (Lo-Hi)
20	DesiredAccessRights	word (Lo-Hi)
22	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	OpenCreateAction	byte
13	OCRetFlags	byte
14	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Parameters

Namespace

(Request) Specifies the name space (see “**Namespace Values**” on page 636).

## OpenCreateMode

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

## SearchAttributes

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

## ReturnInfoMask

(Request) Specifies the information you want returned about a file or subdirectory.

## CreateAttributes

(Request) Specifies the DOS name space attributes.

## DesiredAccessRights

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

## EnhNWHandlePathStruct

(Request) Specifies the [EnhNetWareHandlePathStruct](#) (page 592), which is used to pass the file/subdirectory handle or path.

## FileHandle

(Reply) Specifies the file handle of the file being opened or created.

## OpenCreateAction

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

## OCRetFlags

(Reply) Specifies a callback field that can contain two values:

- 1 Indicates that this request has been registered for callback if someone else tries to open the file.
- 0 Indicates that no callback has been registered.

## NetWareInfoStruct

(Reply) Points to the [NetWareInformationStructure](#), which contains information such as the creation date/time, attributes, inherited rights, and name space.

## EnhNetWareFileNameStruct

(Reply) Points to the [EnhNetWareFileNameStruct](#) (page 591), which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory with Callback** is an enhancement of **Open/Create File or Subdirectory** 0x2222 89 01. On its return, the OCRetFlags (OpenCallBackReturnFlags) field in the reply structure notifies you of completion. Subdirectories can be created, but not opened, by the client.

# Open/Create File or Subdirectory II with Callback 0x2222 89 33

Creates or opens the file and sets a callback flag.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (33)	byte
8	Namespace	byte
9	DataStream	byte
10	OpenCreateMode	byte
11	Reserved	byte
12	Search Attributes	word (Lo-Hi)
14	Reserved	word (Lo-Hi)
16	ReturnInfoMask	long (Lo-Hi)
20	CreateAttributes	long (Lo-Hi)
24	DesiredAccessRights	word (Lo-Hi)
26	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	FileHandle	byte[4] (Hi-Lo)
12	OpenCreateAction	byte
13	OCRetFlags	byte
14	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Parameters

### NameSpace

(Request) Specifies the name space (see “[NameSpace Values](#)” on page 636).

### DataStream

(Request) Specifies the data stream number if the name space is Macintosh (see “[DataStream Values](#)” on page 633).

### OpenCreateMode

(Request) Specifies whether you are creating a new file, replacing a current file, or opening a current file (see “[OpenCreateMode Values](#)” on page 636).

### SearchAttributes

(Request) Specifies the create and open options (see “[SearchAttributes Values](#)” on page 639).

### ReturnInfoMask

(Request) Specifies the information you want returned about a file or subdirectory.

### CreateAttributes

(Request) Specifies the DOS name space attributes.

### DesiredAccessRights

(Request) Specifies the access rights of the file that is being created (see “[DesiredAccessRights Values](#)” on page 633).

### FileHandle

(Reply) Specifies the file handle of the file being opened or created.

### OpenCreateAction

(Reply) Specifies the type of action that was taken regarding the file or subdirectory (see “[OpenCreateAction Values](#)” on page 636).

### OCRetFlags

(Reply) Specifies a callback field that can contain two values:

- |   |  |
|---|--|
| 1 | Indicates that this request has been registered for callback if someone else tries to open the file. |
| 0 | Indicates that no callback has been registered.  |

### NetWareInfoStruct

(Reply) Points to the NetWareInformationStructure.

### EnhNetWareFileNameStruct

(Reply) Points to the [EnhNetWareFileNameStruct](#) (page 591), which contains the name and length for the file or subdirectory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

**Open/Create File or Subdirectory II with Callback** is an enhancement of **Open/Create File or Subdirectory** 0x2222 89 30. On its return, the OCRetFlags (OpenCallBackReturnFlags) field in the reply structure notifies you of completion. Subdirectories can be created, but not opened, by the client.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct](#) (page 592).

# Recover Salvageable File 0x2222 89 17

Recovers a file or subdirectory entry that was found by calling **Scan Salvageable Files 0x2222 89 16 (page 559)**.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (17)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	ScanSequence	long (Lo-Hi)
14	ScanVolume	long (Lo-Hi)
18	ScanDirectoryBase	long (Lo-Hi)
22	DataTypeFlag	byte
23	NewFileNameLen	byte
24/25	NewFileName	byte[NewFileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

DataTypeFlag indicates what format the file name in the NewFileName field is in:

- 0 ASCII
- 1 UTF8

By default, the length-preceding field is a byte. If the data is in UTF8 format, the length-preceding field is a WORD (two bytes) in Lo-Hi order.

You cannot use file names or wildcard characters to search for salvageable files or subdirectories.

ScanSequence is the value that was used in the **Scan Salvageable Files** request.

# Rename Or Move a File or Subdirectory 0x2222 89 04

Renames or moves a file or subdirectory.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (04)	byte
8	NameSpace	byte
9	<b>RenameFlag</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	SrcEnhNWHandlePathS1	structure
19	DstEnhNWHandlePathS2	structure
26	SrcDstPathStrings[]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

SrcEnhNWHandlePathS1 and DstEnhNWHandlePath2 are of type **EnhNetWareHandlePathStruct** (page 592) except that the PathInfo array is not contained in these structures (which are 13 bytes in length).

ComponentCount in these two structures is used to specify how many components exist for each structure.

SrcDstPathStrings contains the source component path (if present) and the destination component path (if present).



# Scan File or Subdirectory for Trustees 0x2222 89 05

Scans a file or subdirectory for trustees.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (05)	byte
8	NameSpace	byte
9	MaxReplyObjectIDCount	byte
10	SearchAttribute	word (Lo-Hi)
12	SearchSequence	long (Lo-Hi)
16	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence	long (Lo-Hi)
12	ObjectIDCount	word (Lo-Hi)
14	TrusteeStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Set SearchSequence to zero on the first request. For each subsequent request, set SearchSequence to the returned NextSearchSequence value. When NextSearchSequence is -1, you have received all the trustees.

The server sends as many trustee structures as specified by MaxReplyObjectIDCount, or as many as fit in the reply buffer.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#) .

# Scan Salvageable Files 0x2222 89 16

Scans a subdirectory for any files that have been deleted but not yet purged.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (16)	byte
8	NameSpace	byte
9	DataStream	byte
10	ReturnInfoMask	long (Lo-Hi)
14	ScanSequence	long (Lo-Hi)
18	EnhNWHandlePathStruct	structure

## Reply Format

Offset	Content	Type
Reply header		
8	NextScanSequence	long (Lo-Hi)
12	DeleteTime	word (Lo-Hi)
14	DeleteDate	word (Lo-Hi)
16	DeletorID	long (Hi-Lo)
20	ScanVolume	long (Lo-Hi)
24	ScanDirectoryBase	long (Lo-Hi)
28	NetWareInfoStruct	structure
xx	EnhNetWareFileNameStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

EnhNWHandlePathStruct must point to a subdirectory path. No file names or wildcards are allowed in the search string. EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct](#) (page 592).

Set ScanSequence to -1 on the first request. For each subsequent request, use the returned NextScanSequence value.

# Search for File or Subdirectory 0x2222 89 03

Searches for a file or subdirectory, starting with the SearchSequence number returned by the **Initialize Search** request.

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (03)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	SearchSequence[9]	byte
25	DataTypeFlag	byte
26	SearchPatternLen	byte (ASCII) or WORD (Lo-Hi for UTF8)
27/28	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	Reserved (0)	byte
18	NetWareInfoStruct	structure
xx	<b>EnhNetWareFileNameStruct</b>	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The length field in the EnhNetWareFileNameStruct is determined by the data type flag that is passed in the request. If the data flag specifies an ASCII format, the length is 1 byte. If the data type specifies a UTF8 format, the length is 2 bytes (WORD).

EnhNetWareFileNameStruct is always returned when you call **Search for File or Subdirectory**. With all other 89 requests, however, this field is not returned unless the Include File Name Structure bit is set in ReturnInfoMask.

You can iteratively call **Search for a File**. On the first request, set LastSearchIndex to 0xFFFF. For subsequent requests, set LastSearchIndex to the NextSearchIndex returned by the previous request. You can then search for all files that match the specified search pattern. (Wildcard characters are supported.)

## See Also

**Initialize Search 0x2222 89 02 (page 538)**

# Search for File or Subdirectory Set 0x2222 89 20

Searches for a file or subdirectory, starting with the SearchSequence number returned by [Initialize Search 0x2222 89 02 \(page 538\)](#).

**NetWare Servers:** 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (20)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	<b>SearchAttributes</b>	word (Lo-Hi)
12	<b>ReturnInfoMask</b>	long (Lo-Hi)
16	ReturnInfoCount	word (Lo-Hi)
18	SearchSequence[9]	byte
25	DataTypeFlag	byte
26	SearchPatternLen	byte (ASCII) or WORD (Lo-Hi for UTF8)
27/28	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	MoreEntriesFlag	byte
18	InfoCount	word (Lo-Hi)
20	<b>NetWareInformationStruct</b>	structure
xx	<b>EnhNetWareFileNameStruct</b>	structure
yy	...	

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Search for File or Subdirectory Set** returns as many NetWareInfoStructs as are requested by ReturnInfoCount (or that fit in a packet). InfoCount is the count of NetWareInfoStructs that are returned.

DataTypeFlag indicates what format the file name in the NewFileName field is in:

- 0 ASCII
- 1 UTF8

By default, the length-preceding field is a byte. If the data format is UTF8, the length-preceding field is a WORD (two bytes) in Lo-Hi order.

MoreEntriesFlag is set to -1 (0xFF) when more entries are available. It is set to zero when there are no more entries.

To get the NetWareFileNameStruct, IncludeFileNameStruct must be set in ReturnInfoMask.

For example, **Search for File or Subdirectory Set** might return the following:

```
InfoCount = 2
IncludeFileNameStruct = TRUE

0   NetWareInfoStruct
+   NetWareFileNameStruct
+   NetWareInfoStruct
+   NetWareFileNameStruct
```



# Search for File or Subdirectory Set (Extended Errors) 0x2222 89 40

Searches for a file or subdirectory, starting with the SearchSequence number returned by Initialize Search.

NetWare Servers: 6.5 SP2

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (40)	byte
8	NameSpace	byte
9	DataStream	byte
10	SearchAttributes	word (Lo-Hi)
12	ReturnInfoMask	long (Lo-Hi)
16	ReturnInfoCount	word (Lo-Hi)
18	SearchSequence[9]	byte
25	DataTypeFlag	byte
26	SearchPatternLen	byte (ASCII) or WORD (Lo-Hi for UTF8)
27/28	SearchPattern	byte[SearchPatternLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextSearchSequence[9]	byte
17	MoreEntriesFlag	byte
18	InfoCount	word (Lo-Hi)
20	NetWareInformationStruct	structure
xx	EnhNetWareFileNameStruct	structure
yy	...	

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Search for File or Subdirectory Set (Extended Errors)** returns extended error codes. If the path ends with a subdirectory that doesn't exist, `ERR_INVALID_PATH` (0x899C) is returned, rather than 0x89FF. If a file exists but you have insufficient rights to see it, `ERR_ACCESS_DENIED` (0x89A8) is returned, rather than 0x89FF. To access these extended error codes, the operating system ORs 0x80000000 to the search attributes so that the extended error codes can be returned. You do not have to pass this bit.

**Search for File or Subdirectory Set (Extended Errors)** returns as many `NetWareInfoStructs` as are requested by `ReturnInfoCount` (or that fit in a packet). `InfoCount` is the count of `NetWareInfoStructs` that are returned.

`DataTypeFlag` indicates what format the file name in the `SearchPattern` field is in:

0 ASCII

1 UTF8

If the data type indicates ASCII format, the length field is 1 byte. If the data type indicates UTF8 format, the length field is two bytes (WORD).

`MoreEntriesFlag` is set to -1 when more entries are available. It is set to zero when there are no more entries.

To get the `EnhNetWareFileNameStruct`, the `IncludeFileNameStruct` must be set in `ReturnInfoMask`.

For example, **Search for File or Subdirectory Set** might return the following:

```
InfoCount = 2
IncludeFileNameStruct = TRUE
```

```
0 NetWareInfoStruct
+ NetWareFileNameStruct
+ NetWareInfoStruct
+ NetWareFileNameStruct
```

## Set Short Directory Handle 0x2222 89 09

Sets a short directory handle.

**NetWare Servers:** 6.5 SP2

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (89)	byte
7	SubFunction (09)	byte
8	NameSpace	byte
9	<b>DataStream</b>	byte
10	DstDirHandle	byte
11	Flags	byte
12	EnhNWHandlePathStruct	structure

### Reply Format

Offset	Content	Type
Flags Value		
8	Volume	long (Lo-Hi)
12	DirectoryBase	long (Lo-Hi)
16	DOSDirectoryBase	long (Lo-Hi)
20	<b>EnhNetWareFileNameStruct</b>	long (Lo-Hi)

### Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If the Flags field is set to 0x40, the reply contains the four items listed in the Reply Format section.  
If the input Flags field is zero, no data is returned.

EnhNWHandlePathStruct is of type [EnhNetWareHandlePathStruct \(page 592\)](#).

# 23 Structures

This section describes the File System structures and their fields.

# 64BitFileSizeStruct

Contains the 64-bit file size of a file on an NSS volume.

## Syntax

```
typedef struct
{
    UINT64    FileSize;
} 64BitFileSizeStruct;
```

## Fields

### FileSize

Specifies the 64-bit file size of a file on an NSS volume. If the request is made on a traditional volume, a 32-bit file size is returned with the upper four bytes zeroed.

# AllAttrStruc

Contains information about the number of attributes for a scanned item, along with [AllDirBaseNumStruc](#).

## Syntax

```
typedef struct
{
    LONG    NumberOfAttributes;
    LONG    Attributes[NumberOfAttributes];
} AllAttrStruc;
```

## Remarks

NumberOfAttributes depends on the number of Name Spaces loaded on the requested volume.

# AllDirBaseNumStruc

Contains attribute information for a scanned item, along with [AllAttrStruc](#).

## Syntax

```
typedef struct
{
    LONG    NumberOfDirBaseNumbers;
    LONG    DirectoryBaseNumbers[NumberOfDirBaseNumbers];
} AllDirBaseNumStruc;
```

## Remarks

NumberOfDirBaseNumbers depends on the number of Name Spaces loaded on the requested volume.



# ArchiveInfoStruct

Contains the archived date, time, and archiver ID information.

## Syntax

```
typedef struct
{
    WORD    ArchivedTime; (Lo-Hi)
    WORD    ArchivedDate; (Lo-Hi)
    LONG    ArchiverID; (Hi-Lo)
} CreationInfoStruct;
```

# AttributesStruct

Contains information about the attributes and flags.

## Syntax

```
typedef struct
{
    LONG    Attributes; (Lo-Hi)
    WORD    Flags; (Lo-Hi)
} AttributesStruct;
```

# CreationInfoStruct

Contains the creation date, time, and owner information.

## Syntax

```
typedef struct
{
    WORD    CreationTime; (Lo-Hi)
    WORD    CreationDate; (Lo-Hi)
    LONG    CreatorID; (Hi-Lo)
} CreationInfoStruct;
```

# CreatorStruc

Contains information about the creator name space for the scanned item.

## Syntax

```
typedef struct
{
    LONG    CreatorNameSpace;
    BYTE    CreatorNameLen;
    BYTE    CreatorName[CreatorNameLen];
} CreatorStruc;
```

# DataStreamFATInfo

Contains the number of FAT blocks associated with a file.

## Syntax

```
typedef struct
{
    LONG    DataStreamNumber;
    LONG    DataStreamFATBlockSize;
} DataStreamFATInfo;
```

# DataStreamSizeInfo

Contains the data stream size information.

## Syntax

```
typedef struct
{
    LONG    DataStreamNumber;
    LONG    DataStreamSize;
} DataStreamSizeInfo;
```

# DataStreamSizeStruct

Contains the data stream size information.

## Syntax

```
typedef struct
{
    LONG    DataStreamSize; (Lo-Hi)
} DataStreamSizeStruct;
```

## Remarks

DataStreamSize is synonymous to the file size in the DOS name space.

# DataStreamSizesStruc

Contains data stream information for a scanned file.

## Syntax

```
typedef struct
{
    LONG    NumberOfSizes;
    LONG    Sizes[NumberOfSizes];
} DataStreamSizesStruc;
```

## Remarks

Sizes is dependent on the name space.



# DirDiskSpaceResList

Contains information about the directory path restrictions.

## Syntax

```
typedef struct DirDiskSpaceResList
{
    BYTE    LevelNumber;
    LONG    MaximumAmount;
    LONG    CurrentAmount;
};
```

## Fields

LevelNumber

Specifies the distance from the directory to the root.

MaximumAmount

Specifies the maximum amount of space assigned to a directory.

CurrentAmount

Specifies the amount of space assigned to a directory (minus the amount of space used by a directory and its subdirectories).

# DirEntryStruct

Contains the directory entry information.

## Syntax

```
typedef struct
{
    LONG    DirectoryEntryNumber; (Lo-Hi)
    LONG    DOSDirectoryEntryNumber; (Lo-Hi)
    LONG    VolumeNumber; (Lo-Hi)
} DirEntryStruct;
```

# DOSNameStruct

Contains information about the DOS name for a scanned item.

## Syntax

```
typedef struct
{
    BYTE    DOSNameLength;
    BYTE    DOSName[DOSNameLength];
} DOSNameStruct;
```

# DOSTimeInformationBlock

Contains the DOS timestamp information.

## Syntax

```
typedef struct
{
    LONG    CreateDateAndTime; (Lo-Hi)
    LONG    LastArchiveDateAndTime; (Lo-Hi)
    LONG    LastUpdatedDateAndTime; (Lo-Hi)
    LONG    LastAccessedDate; (Lo-Hi)
    LONG    LastModifiedInSeconds; (Lo-Hi)
} DOSTimeInformationBlock;
```

## Remarks

LastModifiedInSeconds contains the second relative to the year 2000.

# DSSpaceAllocateStruct

Contains information about the space allocated for the data stream.

## Syntax

```
typedef struct
{
    LONG    DataStreamSpaceAlloc; (Lo-Hi)
} DSSpaceAllocateStruct;
```

# DStreamActual

Contains the list of data streams and the number of FAT blocks associated with a file.

## Syntax

```
typedef struct
{
    LONG                NumberOfDataStreams;
    DataStreamFATInfo  ds[NumberOfDataStreams];
} DStreamActual;
```

# DStreamLogical

Contains the list of data streams and their sizes on a file.

## Syntax

```
typedef struct
{
    LONG                NumberOfDataStreams;
    DataStreamSizeInfo ds[NumberOfDataStreams];
} DStreamLogical;
```

# DataStreamSizeInfo

Contains the data stream size information.

## Syntax

```
typedef struct
{
    LONG    DataStreamNumber;
    LONG    DataStreamSize;
} DataStreamSizeInfo;
```



# EAttrInfoStruct

Contains the extended attributes information.

## Syntax

```
typedef struct
{
    LONG    ExtAttrDataSize; (Lo-Hi)
    LONG    ExtAttrCount; (Lo-Hi)
    LONG    ExtAttrKeySize; (Lo-Hi)
} EAttrInfoStruct;
```

# EffectiveRightsStruct

Returns the effective rights of the specified path.

## Syntax

```
typedef struct  
{  
    LONG    EffectiveRights;  
} EffectiveRightsStruct;
```

# EnhNetWareFileNameStruct

Passes a file name and its length back to the client.

## Syntax

```
struct
{
    BYTE    FileNameLength;
    BYTE    FileName[];
};
```

## Fields

FileNameLength

Specifies the length of the file name. If the request indicates an ASCII data type, this field is 1 byte long. If the request indicates a UTF8 data type, this field is 2 bytes (WORD) long (Lo-Hi).

FileName

Specifies the name of the file. To allow a UTF8 formatted string to be represented, the name of the file can be a maximum of 255 bytes in ASCII format and 768 bytes in UTF8 format.

## Remarks

EnhNetWareFileNameStruct is always returned to **Search For File or Subdirectory** 0x2222 89 03. For other requests, the IncludeFileNameStructure bit needs to be set in ReturnInfoMask before this structure will be returned.

# EnhNetWareHandlePathStruct

Contains information about a handle or path.

## Syntax

```
struct
{
    LONG        DirectoryBaseOrShortHandle; (Lo-Hi)
    BYTE        VolumeNumber;
    BYTE        HandleFlag;
    BYTE        DataTypeFlag;
    BYTE        Reserved[5];
    BYTE        PathComponentCount;
    STRUCT      Pcomponent [PathComponentCount];
};
```

## Fields

### HandleFlag

Specifies whether the handle is a short directory handle:

0x00 ShortDirectoryHandle  
0x01 DirectoryBase  
0xFF NoHandlePresent

### DataTypeFlag

Specifies the format of the path string:

0x00 ASCII\_DATA\_TYPE  
0x01 UTF8\_DATA\_TYPE

### PathComponentCount

Specifies the count for each directory and subdirectory. For example, sys:test\nlm\_test would be a path component count of three.

## Remarks

The only difference between EnhNetWareHandlePathStruct and **NetWareHandlePathStruct** (page 609) is that the enhanced version has a DataTypeFlag field that specifies whether the path is in ASCII or UTF8 format.

The client can pass a handle or path by using the volume number, directory base, or directory handle field.

The first part of EnhNetWareHandlePathStruct is used to pass either a 1-byte directory handle (NetWare 2.15 short style) or a 4-byte directory base (NetWare 3.1 style).

The second portion of NetWareHandlePathStruct is used to pass a qualified NetWare Path (Len, String) in component form. The length field is determined by the data type flag that is passed in the NCP request. If the data type indicates an ASCII path, the length is 1 byte. If the data type indicates a UTF8 path, the length is 2 bytes (WORD).

The client can pass a short directory handle (with or without a path), a directory base (with or without a path), or a fully qualified path. The server always checks `HandleFlag` to see if the handle is a short directory handle (0), a directory base (1), or No Handle Present (FF). If the server finds that you have passed a short directory handle or a directory base, it can use `PathComponentCount`. If the server find that the client hasn't passed a handle, the server expects the first path component to contain the volume name. If there are additional components, the server handles them after generating a volume number.

## Example

The following is an example of how `HandleFlag` is used:

```
if (HandleFlag == DirectoryBase)
{
    /* Client passed in Volume Number and Directory Base */
    /* Set path base from directory base */
}
else
{
    if (HandleFlag == NoHandlePresent)
    {
        /* Generate Volume Number from the first path component */
        /* Set path base to 0, and subtract 1 from path count */
    }
    if (HandleFlag == ShortDirectoryHandle)
    {
        /* Get path base from allocated short directory handle */
        /* Get volume number from directory handle table entry */
    }
    /* check to see if more components are present */
    if (PathCount > 1)
    {
        /* find last component */
        /* map path to last component */
        /* set the path base */
    }
}
/* return updated volume number and path base */
```

# FlushTimeStruct

Contains information about the flush time for the scanned item.

## Syntax

```
typedef struct
{
    LONG    FlushTime;
} FlushTimeStruct;
```

# IDStruc

Contains information about the IDs for the scanned item.

## Syntax

```
typedef struct
{
    LONG    OwnerID;
    LONG    LastArchivedID;
    LONG    LastUpdatedID; /* valid only for files */
} IDStruc;
```

# InfoBlock

Contains the requested information for each scanned item.

## Syntax

```
typedef struct
{
    InfoCCode    ccode;
} InfoBlock;
```

## Remarks

If completionCode of InfoCCode is zero, the requested information follows.



# InfoCCode

Contains information about the scanned item.

## Syntax

```
typedef struct
{
    LONG    completionCode;
    LONG    FolderFlag;
    LONG    DirectoryBaseNumber;
} InfoBlock;
```

## Fields

completionCode

Specifies the completion code for the scanned item.

FolderFlag

Specifies whether the item is a folder.

DirectoryBaseNumber

Specifies the name space.

# LastAccessedTimeStruct

Contains the last time a file was accessed.

## Syntax

```
typedef struct
{
    WORD    LastAccessedTime;
} LastAccessedTimeStruct;
```

## Fields

LastAccessedTime

Specifies the last time a file was accessed, which includes being opened, written to, or having its time/date stamp changed.

# LastUpdatedInSecondsStruct

Contains the last time a file or directory was updated, relative to the year 2000.

## Syntax

```
typedef struct
{
    LONG    SecondsRelativeToTheYear2000;
} LastUpdatedInSecondsStruct;
```

# limb

## Syntax

```
typedef struct
{
    long    limbFlags; (Lo-Hi)
    long    limbVolumeNumber; (Lo-Hi)
    long    limbDirectoryBase; (Lo-Hi)
    long    limbScanNumber; (Lo-Hi)
    long    limbNameSpace; (Lo-Hi)
} limb;
```

## Fields

limbScanNumber

Is used only if ScanEntireFolder is set.

## Remarks

limbFlags can have the following values:

- 0x00000002 Scan Entire Folder (wild search)
- 0x00000004 Scan Files Only (valid only if Scan Entire Folder is set)
- 0x00000008 Scan Folders Only (valid only if Scan Entire Folder is set)
- 0x00000010 Allow System Files/Folders
- 0x00000020 Allow Hidden Files/Folders

# MacFinderInfoStruct

Contains the MAC finder information for a scanned item.

## Syntax

```
typedef struct
{
    BYTE    MacFinderInfo[32];
} MacFinderInfoStruct;
```

# MacTimeStruct

Contains information about the MAC time for the scanned item.

## Syntax

```
typedef struct
{
    LONG    MACCreateTime;
    LONG    MACBackupTime;
} MacTimeStruct;
```

## Fields

MACCreateTime

Specifies the time the Macintosh entry was created.

MACBackupTime

Specifies the time that the Macintosh entry was last backed up.

# MaxSpaceStruc

Contains information about data stream sizes for a scanned folder.

## Syntax

```
typedef struct
{
    LONG    NumberOfSizes = 1;
    LONG    MaximumSpaceRestriction;
} MaxSpaceStruc;
```

# ModifyDOSInfoStructure

Contains DOS-specific information in file or subdirectories when the default name space is not DOS.

## Syntax

```
struct
{
    WORD    FileAttributes; (Lo-Hi)
    BYTE    FileMode;
    BYTE    FileXAttributes;
    WORD    CreationDate; (Lo-Hi)
    WORD    CreationTime; (Lo-Hi)
    LONG    CreatorsID; (Hi-Lo)
    WORD    ModifiedDate; (Lo-Hi)
    WORD    ModifiedTime; (Lo-Hi)
    LONG    ModifiersID; (Hi-Lo)
    WORD    ArchivedDate; (Lo-Hi)
    WORD    ArchivedTime; (Lo-Hi)
    LONG    ArchiversID; (Hi-Lo)
    WORD    LastAccessDate; (Lo-Hi)
    WORD    InheritanceGrantMask; (Lo-Hi)
    WORD    InheritanceRevokeMask; (Lo-Hi)
    LONG    MaximumSpace; (Lo-Hi)
};
```

## Fields

### MaximumSpace

Specifies the user disk space restriction that may have been enabled by an administrator and might be in place for a given user.

## Remarks

ModifyDOSInfoMask tells the server what information in the ModifyDosInfoStructure is valid and is defined as follows:

- 0 Unused Bit
- 1 Attributes Bit
- 2 Creation Date Bit
- 3 Creation Time Bit
- 4 Creator ID Bit
- 5 Archive Date Bit
- 6 Archive Time Bit
- 7 Archiver ID Bit
- 8 Modify Date Bit
- 9 Modify Time Bit
- 10 Modifier ID Bit
- 11 Last Access Bit
- 12 Inheritance Bit



13 Max Space Bit  
14 reserved  
16-31 reserved

# ModifyInfoStruct

Contains the modified date, time, and modifier ID information.

## Syntax

```
typedef struct
{
    WORD    ModifiedTime; (Lo-Hi)
    WORD    ModifiedDate; (Lo-Hi)
    LONG    ModifierID; (Hi-Lo)
    WORD    LastAccessDate; (Lo-Hi)
} ModifyInfoStruct;
```

# NameStruc

Contains information about the name for a scanned item.

## Syntax

```
typedef struct
{
    BYTE    NameLen;
    BYTE    Name[NameLen];
} NameStruc;
```

# NetWareFileNameStruct

Contains information about a file name.

## Syntax

```
struct
{
    BYTE    FileNameLength;
    BYTE    FileName[];
};
```

## Remarks

The file name is limited to a maximum of 255 characters.

NetWareFileNameStruct is always returned to **Search For File or Subdirectory** 0x2222 87 03. For other requests, the IncludeFileNameStructure bit needs to be set in ReturnInfoMask before this structure will be returned.

# NetWareHandlePathStruct

Contains information about a handle or path.

## Syntax

```
struct
{
    BYTE    VolumeNumber;
    LONG    DirectoryBaseOrShortHandle; (Lo-Hi)
    BYTE    HandleFlag;
    BYTE    PathComponentCount;
    STRUC   Pcomponent [PathComponentCount];
};
```

## Fields

### HandleFlag

Specifies whether the handle is a short directory handle:

0x00 ShortDirectoryHandle  
0x01 DirectoryBase  
0xFF NoHandlePresent

### PathComponentCount

Specifies the count for each directory and subdirectory. For example, sys:test\nlm\_test would be a path component count of three.

### Pcomponent

Specifies Pcomponent, which contains the count of the number of components.

## Remarks

NetWareHandlePathStruct totals 307 bytes. If you have other structures or data that follow this structure, that data will start after the 307 bytes that are reserved for NetWareHandlePathStruct.

The first part of NetWareHandlePathStruct is used to pass a 1-byte short directory handle (NetWare 2.15 style) or a 4-byte directory base (NetWare 3.1 style).

The second portion of NetWareHandlePathStruct is used to pass a qualified NetWare Path (Len, String) in component form.

You can pass a short directory handle (with or without a path), a directory base (with or without a path), or a fully qualified path. The server always checks HandleFlag to see if the handle is a short directory handle (0), a directory base (1), or No Handle Present (FF). If the server finds that you have passed a short directory handle or a directory base, it can use Pcomponent. If the server finds that you haven't passed a handle, the server expects the first path component to contain the volume name. If there are additional components, the server handles them after generating a volume number.

Rename File/Subdirectory uses a slightly modified NetWareHandlePathStruct.

## Example

The following is an example of how HandleFlag is used:

```
#define ShortDirectoryHandle 0x00
#define DirectoryBase 0x01
#define NoHandlePresent 0xFF

if (HandleFlag == DirectoryBase)
{
    /* Client passed in Volume Number and Directory Base */
    /* Set path base from directory base */
}
else
{
    if (HandleFlag == NoHandlePresent)
    {
        /* Generate Volume Number from the first path component */
        /* Set path base to 0, and subtract 1 from path count */
    }
    if (HandleFlag == ShortDirectoryHandle)
    {
        /* Get path base from allocated short directory handle */
        /* Get volume number from directory handle table entry */
    }
    /* check to see if more components are present */
    if (PathCount > 1)
    {
        /* find last component */
        /* map path to last component */
        /* set the path base */
    }
}
/* return updated volume number and path base */
```

# NetWareInformationStruct

Contains information corresponding to the bits set in ReturnInfoMask.

## Syntax

```
DSSpaceAllocateStruct (Data Stream Alloc Bit)
AttributesStruct (Attributes Bit)
DataStreamSizeStruct (Data Stream Size Bit)
TotalStreamSizeStruct (Total Stream Size Bit)
CreationInfoStruct (Creation Bit)
ModifyInfoStruct (Modify Bit)
ArchiveInfoStruct (Archive Bit)
RightsInfoStruct (Rights Bit)
DirEntryStruct (Directory Entry Bit)
EAInfoStruct (Extended Attribute Bit)
NSInfoStruct (Name Space Bit)
```

## Remarks

NetWareInformationStruct defines the way you request information from a server and the way the server responds back to you. When you set a bit in **ReturnInfoMask**, the server uses the corresponding structure, within NetWareInformationStruct, to return the requested information in a reply packet. For example, if the Rights Bit of the ReturnInfoMask field is set, the server uses the RightsInfoStruct field of NetWareInformationStruct to return information about a specified file's rights.

The server always returns a full NetWareInformationStruct no matter which bits you set; however, NetWareInformationStruct will contain valid information only for those fields that you set and only upon successful completion of the request.

You should request only the information you need since the more information you request, the longer NetWareInformationStruct takes to return.

DataStreamSpaceAllocatedInformation takes unusually long time to return.

# NetWareTrusteeStructure

Contains information about an object and its trustee rights.

## Syntax

```
struct
{
    LONG    ObjectID; (Hi-Lo)
    WORD    TrusteeRights; (Lo-Hi)
};
```

## Remarks

The following NetWare 3.1 requests use this structure:

**Add Trustee Set to File or Subdirectory 0x2222 87 10 (page 289)**

**Delete Trustee Set from File or SubDirectory 0x2222 87 11 (page 316)**

**Scan File or Subdirectory for Trustees 0x2222 87 05 (page 468)**



# NSAttributeStruct

Contains the file attribute value of the requested entry.

## Syntax

```
typedef struct
{
    LONG    FileAttributes;
} NSAttributeStruct;
```

# NSInfoStruct

Contains the name space number of the creator.

## Syntax

```
typedef struct
{
    LONG    CreatorNameSpaceNumber; (Lo-Hi)
} NSInfoStruct;
```

# ParentBaseIDStruct

Contains the parent directory base number for a file or subdirectory.

## Syntax

```
typedef struct
{
    LONG    ParentBaseID;
} ParentBaseIDStruct;
```

# PathComponent

## Syntax

```
struct PathComponent
{
    byte    pathLength;
    byte    string[];
};
```

# PathCookie

## Syntax

```
struct PathCookie
{
    word    Flags; (Lo-Hi)
    long    Cookie1; (Lo-Hi)
    long    Cookie2; (Lo-Hi)
};
```

## Remarks

Cookie1 and Cookie2 are initially set to -1. When Cookie2 comes back in the reply as -1, the path components have been completely transferred over.

If bit 0 of Flags is set to 1, the last component is a file name.

# Pcomponent

Contains information about a qualified NetWare path.

## Syntax

```
struct
{
    BYTE    PathNameLen;
    BYTE    PathName[];
} Pcomponent[];
```

## Remarks

Pcomponent is used to pass a qualified NetWare path in component form.

# ReferenceCount

Contains the reference count for the scanned item.

## Syntax

```
typedef struct
{
    LONG    ReferenceCount;
} ReferenceCountStruc;
```

# ReferenceIDStruct

Contains the change count information on a file, directory, or volume.

## Syntax

```
typedef struct
{
    WORD    CurrentReferenceID;
} ReferenceIDStruct;
```

## Remarks

ReferenceIDStruct can be used to tell if something changed.



# RightsInfoStruct

Contains the inherited rights mask information.

## Syntax

```
typedef struct
{
    WORD    InheritedRightsMask; (Lo-Hi)
} RightsInfoStruct;
```

# ScanInfoFileName

Contains the file entry number for a salvageable file.

## Syntax

```
typedef struct ScanInfoFileName
{
    LONG    SalvageableFileEntryNumber;
    BYTE    FileNameLength;
    BYTE    FileName[FileNameLength];
};
```

# ScanInfoFileNoFileName

Contains the file information for a salvageable file.

## Syntax

```
typedef struct ScanInfoNoFileName
{
    LONG    SalvageableFileEntryNumber;
};
```

# SiblingCountStruct

Contains the number of siblings in a subdirectory.

## Syntax

```
typedef struct
{
    LONG    SiblingCount;
} SiblingCountStruct;
```

# TotalStreamSizeStruct

Contains the data stream size information.

## Syntax

```
typedef struct
{
    LONG    TtlDSDskSpaceAlloc; (Lo-Hi)
    WORD    NumberOfDataStreams; (Lo-Hi)
} TotalStreamSizeStruct;
```

## Fields

TtlDSDskSpaceAlloc

Specifies the number of 4K blocks that the specified file is using.

# TrusteeStruct

Contains the trustee information.

## Syntax

```
typedef struct
{
    LONG    ObjectID; (Hi-Lo)
    WORD    TrusteeRights; (Lo-Hi)
} TrusteeStruct;
```

## Fields

ObjectID

Specifies the number of TrusteeStructu entries.

TrusteeRights

Specifies the rights for the specified trustee.

## Remarks

TrusteeStruct is a 6-byte structure.

# VolInfoStructure

## Syntax

```
struct VolInfoStructure
{
    LONG    VolumeType; (Lo-Hi)
    LONG    StatusFlagBits; (Lo-Hi)
    LONG    SectorSize; (Lo-Hi)
    LONG    SectorsPerCluster; (Lo-Hi)
    LONG    VolumeSizeInClusters; (Lo-Hi)
    LONG    FreedClusters; (Lo-Hi)
    LONG    SubAllocFreeableClusters; (Lo-Hi)
    LONG    FreeableLimboSectors; (Lo-Hi)
    LONG    NonFreeableLimboSectors; (Lo-Hi)
    LONG    NonFreeableAvailableSubAllocSectors; (Lo-Hi)
    LONG    NotUsableSubAllocSectors; (Lo-Hi)
    LONG    SubAllocClusters; (Lo-Hi)
    LONG    DataStreamsCount; (Lo-Hi)
    LONG    LimboDataStreamsCount; (Lo-Hi)
    LONG    OldestDeletedFileAgeInTicks; (Lo-Hi)
    LONG    CompressedDataStreamsCount; (Lo-Hi)
    LONG    CompressedLimboDataStreamsCount; (Lo-Hi)
    LONG    UncompressableDataStreamsCount; (Lo-Hi)
    LONG    PreCompressedSectors; (Lo-Hi)
    LONG    CompressedSectors; (Lo-Hi)
    LONG    MigratedFiles; (Lo-Hi)
    LONG    ClustersUsedByFAT; (Lo-Hi)
    LONG    ClustersUsedByDirectories; (Lo-Hi)
    LONG    ClustersUsedByExtendedDirectories; (Lo-Hi)
    LONG    TotalDirectoryEntries; (Lo-Hi)
    LONG    UnusedDirectoryEntries; (Lo-Hi)
    LONG    TotalExtendedDirectoryExtants; (Lo-Hi)
    LONG    UnusedExtendedDirectoryExtants; (Lo-Hi)
    LONG    ExtendedAttributesDefined; (Lo-Hi)
    LONG    ExtendedAttributeExtantsUsed; (Lo-Hi)
    LONG    DirectoryServicesObjectID; (Lo-Hi)
    LONG    VolumeLastModifiedDateAndTime; (Lo-Hi)
};
```

## Fields

### VolumeType

Specifies the defined type of the current volume:

- 0 VINetWare386
- 1 VINetWare286
- 2 VINetWare386x30
- 3 VINetWare386x31

### StatusFlagBits

Specifies the options that are currently available on the volume:

- 0x01 SubAllocEnableBit

0x02 CompressionEnabledBit  
0x04 MigrationEnableBit  
0x08 AuditingEnabledBit  
0x10 ReadOnlyEnableBit  
0x20 ImmediatePurgeBit  
0x40 64BitFileOffsetsEnabledBit  
0x80 UTF8NCPStringsEnabledBit  
0x80000000 NSSVolumeBit

0x40 indicates whether 64-bit offsets are enabled for a given volume. Only NSS volumes support 64-bit offsets. NCP requests directed at legacy volumes support 32-bit offsets.

0x80 indicates whether a given volume supports incoming NCP requests that contain UTF8 path strings. Only NSS volumes support this feature.

#### SectorSize

Specifies the sector size (in bytes).

#### SectorsPerCluster

Specifies the number of sectors per cluster or block.

#### VolumeSizeInClusters

Specifies the size of the volume (in clusters or blocks).

#### FreedClusters

Specifies the number of clusters or blocks that are currently free for allocation. It does not include space that is currently available from deleted or limbo files, nor space that could be reclaimed from the suballocation file system.

#### SubAllocFreeableClusters

Specifies the space that can be reclaimed from the suballocation file system.

#### FreeableLimboSectors

Specifies the disk space (in sectors) that can be freed from deleted files.

#### NonFreeableLimboSectors

Specifies the disk space (in sectors) that is currently in deleted files and is not aged enough to be classified as freeableLimboClusters.

#### NonFreeableAvailSubAllocSectors

Specifies the space that is available to the suballocation file system but is not freeable to return as sectors.

#### NotUsableSubAllocSectors

Specifies the disk space that is wasted by the suballocation file system. These sectors cannot be allocated by the suballocation system or used as regular sectors.

#### SubAllocClusters

Specifies the disk space being used by the suballocation file system.

#### DataStreamsCount

Specifies the number of data streams for real files that have data allocated to them.



#### LimboDataStreamsCount

Specifies the number of data streams for deleted files that have data allocated to them.

#### OldestDeletedFileAgeInTicks

Specifies the current age of the oldest file (in ticks).

#### CompressedDataStreamsCount

Specifies the number of data streams for compressed real files.

#### CompressedLimboDataStreamsCount

Specifies the number of data streams for compressed deleted files.

#### UnCompressableDataStreamsCount

Specifies the number of data streams that are not compressible (real and deleted).

#### PreCompressedSectors

Specifies the amount of disk space that was allocated to all files before they were compressed (includes "hole" space).

#### CompressedSectors

Specifies the amount of disk space that is used by all compressed files.

#### MigratedFiles

Specifies the number of migrated files.

#### MigratedSectors

Specifies the amount of migrated disk space (in sectors).

#### ClustersUsedByFAT

Specifies the amount of disk space (in clusters or blocks) that is being used by the FAT table.

#### ClustersUsedByDirectories

Specifies the amount of disk space (in clusters or blocks) that is being used by directories.

#### ClustersUsedByExtendedDirs

Specifies the amount of disk space (in clusters or blocks) that is being used by the extended directory space.

#### TotalDirectoryEntries

Specifies the total number of directories that are available on the volume.

#### UnUsedDirectoryEntries

Specifies the total number of directory entries that are not in use on the volume.

#### TotalExtendedDirectoryExtants

Specifies the amount of extended directory space extants (128 bytes each) that are available on the volume.

#### UnUsedExtendedDirectoryExtants

Specifies the amount of extended directory space extants (128 bytes each) that are not in use on the volume.

#### ExtendedAttributesDefined

Specifies the number of extended attributes that are defined on the volume.

#### ExtendedAttributeExtantsUsed

Specifies the number of extended directory extants that are used by the extended attributes.

#### DirectoryServicesObjectID

Specifies the NDS ID for the volume.

#### VolumeLastModifiedDateAndTime

Specifies the last time any file or subdirectory on the volume was modified (as tracked by the OS).

# VolMntStruct

Returns the volume number without the volume name.

## Syntax

```
struct VolMntStruct
{
    LONG    VolumeNumber;
};
```

# VolMntStructWithName

Returns the volume name with the volume number.

## Syntax

```
struct VolMntStructWithName
{
    LONG    VolumeNumber;
    BYTE    VolumeNameLength;
    BYTE    VolumeName[VolumeNameLength];
};
```

# 24 Values

This section describes the File System values.

## ChangeBits Values

DOS *ChangeBits* can have the following values:

- 0x0001 ModifyNameBit
- 0x0002 FileAttributesBit
- 0x0004 CreateDateBit
- 0x0008 CreateTimeBit
- 0x0010 OwnerIDBit
- 0x0020 LastArchivedDateBit
- 0x0040 LastArchivedTimeBit
- 0x0080 LastArchivedIDBit
- 0x0100 LastUpdatedDateBit
- 0x0200 LastUpdatedTimeBit
- 0x0400 LastUpdatedIDBit
- 0x0800 LastAccessedDateBit
- 0x1000 MaxAccessMaskBit
- 0x2000 MaximumSpaceBit

Macintosh *ChangeBits* can have the following values:

- 0x0001 MacModifyNameBit
- 0x0002 MacFinderInfoBit

## DataStream Values

*DataStream* can have the following values:

- 0 Resource fork or DOS
- 1 Data fork

## DesiredAccessRights Values

The following bits are defined for *DesiredAccessRights*:

- 0 Read Only Mode
- 1 Write Only Mode
- 2 Deny Read Mode
- 3 Deny Write Mode
- 4 Compatibility Mode
- 6 File Write Through Mode
- 10 Delete File Close

The following bits are defined for the inherited rights mask for subdirectories:

- 0 Read Existing File Bit
- 1 Write Existing File Bit
- 2 Old Open Existing File Bit
- 3 Create New Entry Bit
- 4 Delete Existing Bit
- 5 Change Access Control Bit
- 6 See Files Bit
- 7 Modify Entry Bit
- 8 Supervisor Privileges Bit

## FileAttributes Values

The following bits are defined for *FileAttributes*:

Bit	Name	Bit Description
0	Read Only Bit	File can be read but not written to.
1	Hidden Bit	File will not be shown in a normal directory listing.
2	System Bit	File will not be shown in a normal directory listing.
3	Execute Only Bit	File can be loaded for execution only; once this bit has been set, it cannot be cleared.
4	Subdirectory Bit	Entry is a subdirectory, not a file.
5	Archive Bit	Bit is set if the file has been changed since it was last backed up.
6	Execute Confirm Bit	
7	Shareable Bit (file only) or Old Private Bit (subdirectories only)	Bit is set if the file can be simultaneously accessed by more than one user.

- 12 Transaction Bit
- 14 Read Audit Bit
- 15 Write Audit Bit
- 16 Immediate Purge Bit
- 17 Rename Inhibit Bit
- 18 Delete Inhibit Bit
- 19 Copy Inhibit Bit
- 20-23 reserved

## InformationMask Values

*InformationMask* can have the following values:

Hex	Description	Type
0x00000001	DOS Time Information	structure Points to <a href="#">DOSTimeInformationBlock</a> (page 584).
0x00000002	Reference Count	long Points to <a href="#">ReferenceCount</a> (page 619).
0x00000004	DOS Attributes	long
0x00000008	IDs	structure Points to <a href="#">IDStruc</a> (page 595).
0x00000010	Data Stream Sizes	structure For files, points to <a href="#">DataStreamSizesStruc</a> (page 580). For folders, points to <a href="#">MaxSpaceStruc</a> (page 603).
0x00000020	Name Space Attribute	long
0x00000040	EA Present Flag	long
0x00000080	All Attributes	long[NumberOfNameSpacesOnVolume] Points to <a href="#">AllAttrStruc</a> (page 571).
0x00000100	All Directory Base Numbers	long[NumberOfNameSpacesOnVolume] Points to <a href="#">AllDirBaseNumStruc</a> (page 572).
0x00000200	Maximum Access Mask	long
0x00000400	Flush Time	long Points to <a href="#">FlushTimeStruct</a> (page 594).
0x00000800	Parent Base ID	long
0x00001000	MAC Finder Info	structure Points to <a href="#">MacFinderInfoStruct</a> (page 601).
0x00002000	Sibling Count	long
0x00004000	Effective Rights	long
0x00008000	MAC Time	structure Points to <a href="#">MacTimeStruct</a> (page 602).
0x00008000	Last Accessed Time	uint16
0x20000000	DOS Name	structure Points to <a href="#">DOSNameStruct</a> (page 583).
0x40000000	Creator Name Space & Name	structure Points to <a href="#">CreatorStruc</a> (page 576).

Hex	Description	Type
0x80000000	Name	structure
		Points to <a href="#">NameStruc (page 607)</a> .

## InheritedRightsMask Values

The following bits are defined for *InheritedRightsMask*:

- 0 Read Existing File Bit
- 1 Write Existing File Bit
- 2 Old Open Existing File Bit
- 3 Create New Entry Bit
- 4 Delete Existing Bit
- 5 Change Access Control Bit
- 6 See Files Bit
- 7 Modify Entry Bit
- 8 Supervisor Privileges Bit
- 9-15 reserved

## NameSpace Values

*NameSpace* can have the following values:

- 0 DOS
- 1 MAC
- 2 NFS
- 3 FTAM
- 4 OS/2

In NetWare 4.11 or later, the OS/2 name space (OS2.NAM) is the LONG name space (LONG.NAM).

## OpenCreateAction Values

The following bits are defined for *OpenCreateAction*:

- 0 File Open
- 1 File/Subdirectory Created
- 2 File Replaced
- 3 File Is Compressed (NetWare 4.11 and above)
- 7 File is Read Only (NetWare 4.11 and above)

If bits 0-2 are set to zero, no action was taken.

## OpenCreateMode Values

*OpenCreateMode* indicates flag bits that have the following meanings:

- 0x01 OPEN\_FILE



0x02 TRUNCATE\_FILE  
0x08 CREATE\_FILE  
0x20 OPEN\_64BIT\_ACCESS (Allows 64-bit access on NSS volumes)

For **Open/Create File or Subdirectory with Callback 0x2222 87 32 (page 404)** or **Open/Create File or Subdirectory II with Callback 0x2222 87 33 (page 407)**, *OpenCreateMode* can also have the following two bits:

0x40 RO\_ACCESS\_OK  
0x80 OPEN\_CALLBACK

In addition, these values can be combined to have the following effects:

0x0 Invalid action  
0x1 Open existing file (file must exist)  
0x2 Open existing file and truncate it, else create a new file (same as 0xB)  
0x3 Open existing file and truncate it (file must exist)  
0x8 Create new file or subdirectory (file or subdirectory cannot exist)  
0x9 Open existing file or create a new file  
0xA Open existing file and truncate it, else create a new file (same 0xB)  
0xB Open existing file and truncate it, else create a new file

For **Open/Create File or Subdirectory 0x2222 89 01 (page 410)**, *OpenCreateMode* has the following bits defined:

0 Action Open  
1 Action Replace  
2 reserved  
3 Action Create  
4 reserved  
5 Action 64 Bit Access Allowed  
6 reserved  
7 reserved

## OpenRights Values

The following bits are defined for *OpenRights*:

0 Read Only Mode  
1 Write Only Mode  
2 Deny Read Mode  
3 Deny Write Mode  
4 Compatibility Mode  
6 File Write Through Mode

## RenameFlag Values

*RenameFlag* can have the following values:

---

0x01	If set, <i>RRenameToMyself</i> allows files to be renamed to its original name. If this bit is not set and an attempt is made to rename the file to itself, 0x8992 is returned.
------	---

---

---

0x02	If set, RCompatabilityFlag allows files that are marked read only to be opened with a read/write access request. Some application require that this bit be set. This bit is also used to determine whether a file is currently locked or in use.
0x04	If set, RNameOnlyFlag renames only the name space entry name (of the name space passed to this request) and ignores other name entries in other name spaces.

---

## ReturnInfoMask Values

To request information from a server, you need to set the appropriate bits of this mask and send a request to the server. The server uses the corresponding structures for the set bits to return information.

When the Include File Name Structure bit is set, the server returns a file name and a file name length in NetWareFileNameStruct.

Only bits 0-11 are defined in NetWare 3.11; all other bits are defined in NetWare 4.x. The NetWare Compress Information Flag bit must be set to get the NetWare 4.x information.

See [“Extended ReturnInfoMask Values” on page 638](#) for the extended bit definitions.

The following bits are defined for ReturnInfoMask:

- 0 Include File Name Structure
- 1 Data Stream Space Allocated Information (see [DSSpaceAllocateStruct](#))
- 2 Attributes Information (see [AttributesStruct](#))
- 3 Data Stream Size Information (see [DataStreamSizeStruct](#))
- 4 Total Space Allocated For All Data Streams (see [TotalStreamSizeStruct](#))
- 5 Extended Attributes Information (see [EAInfoStruct](#))
- 6 Archive Information (see [ArchiveInfoStruct](#))
- 7 Modify Information (see [ModifyInfoStruct](#))
- 8 Creation Information (see [CreationInfoStruct](#))
- 9 Name Space Information (see [NSInfoStruct](#))
- 10 Directory Information (see [DirEntryStruct](#))
- 11 Rights Information (see [RightsInfoStruct](#))

## Extended ReturnInfoMask Values

See [“ReturnInfoMask Values” on page 638](#) for the basic ReturnInfoMask values.

Bits 16-23 are available only on NetWare 4.11 or above.

If RNewStyle (0x80000000) is set in the ReturnInfoMask, the following extended bits are defined:

- 12 Reference ID (see [ReferenceIDStruct](#))
- 13 Name Space Attribute (see [NSAttributeStruct](#))
- 14 Data Stream Actual (see [DStreamActual](#))
- 15 Data Stream Logical (see [DStreamLogical](#))
- 16 Last Updated in Seconds (see [LastUpdatedInSecondsStruct](#))
- 17 DOS Name (see [DOSNameStruct](#))
- 18 Flush Time (see [FlushTimeStruct](#))
- 19 Parent Base ID (see [ParentBaseIDStruct](#))

- 20 MAC Finder Info (see [MacFinderInfoStruct](#))
- 21 Sibling Count (see [SiblingCountStruct](#))
- 22 Effective Rights (see [EffectiveRightsStruct](#))
- 23 MAC Date & Time (see [MacTimeStruct](#))
- 24 Last Accessed Time (see [LastAccessedTimeStruct](#)) (NetWare 5.0 and above)
- 26 64 Bit File Sizes (see [64BitFileSizeStruct](#)) (NetWare 6.0, Support Pack 2 and above)
- 31 New Style (Compress Reply Info)

The Sibling Count (0x20000000) is always zero for a subdirectory.

## SearchAttributes Values

The following bits are defined for SearchAttributes:

- 1 Hidden Bit
- 2 System Bit
- 4 Subdirectories Only Bit
- 15 All Files and Subdirectories Bit

The system and hidden bits in the client's *SearchAttributes* flag are used to discover or ignore system and hidden files.

If the client sets the Subdirectory bit of *SearchAttributes*, directory information is returned. Otherwise, file information is returned.



# IX

## Message

Message NCPs enable applications to send broadcast messages (1-58 bytes) and pipe message (1-126 bytes) to as many as a hundred specified target connections (workstations). The sending workstation and the target workstation must be attached to the same file server.

Broadcast and pipe messages use file server processing time. For true peer-to-peer communication between applications across the network, applications can use Novell's IPX (Internetwork Packet Exchange) or SPX (Sequenced Packet Exchange) protocols, or NetBIOS. These protocols do not use server processing time and, therefore, promote better performance.

The NetWare utilities SEND, CASTON, and CASTOFF use Message NCPs to send, enable, and prevent broadcast (not pipe) messages.

To access Message NCPs, use the following:

- ♦ [Chapter 25, "Concepts," on page 643](#)
- ♦ [Chapter 26, "NCPs," on page 645](#)



# 25 Concepts

This section explains ideas that are common to Message NCPs.

## Message Buffers

Each file server connection maintains a 58-byte message buffer for broadcast messages and a 6-slot message buffer for pipe messages. Each slot can hold a 126-byte pipe message.

Before sending a pipe message, the sending and receiving connections must establish a message pipe with the Open Message Pipe request.

When one connection sends a message to another connection, the file server places the message in the target connection's message buffer or pipe queue and informs the target connection that a message has arrived. The target connection's shell automatically retrieves the message.

Each connection on a file server has a configurable message mode (00h-03h) that allows the workstation to open or close its message buffer and to enable or disable the automatic message retrieval feature of its shell.





# 26 **NCPs**

This section describes each of the Message NCPs, their Request and Reply formats, and Return Values.

# Broadcast To Console 0x2222 21 09

Broadcasts a message to the system console.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen (2 + MessageLen)	word (Hi-Lo)
9	SubFunctionCode (9)	byte
10	MessageLen	byte
11	Message	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The message can be no longer than 255 bytes and displays at the console on a single line after the ":" prompt.

New messages received at the console overwrite any previous message.

## See Also

**Disable Broadcasts 0x2222 21 02 (page 648), Enable Broadcasts 0x2222 21 03 (page 649)**

# Connection Message Control 0x2222 21 12

Controls Broadcast and watchdog messages for all connections included in the connection list.

NetWare Server: 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (12)	byte
10	ConnectionControlBits	LONG
14	ConnectionListCount	LONG
18	ConnectionList	LONG[ConnectionListCount]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Bad Parameter

## Remarks

*ConnectionControlBits* can have the following values:

- 0x00000001 Enable Broadcast Messages
- 0x00000002 Enable Personal Broadcast Messages
- 0x00000004 Enable Watchdog Messages
- 0x00000010 Disable Broadcast Messages
- 0x00000020 Disable Personal Broadcast Messages
- 0x00000040 Disable Watchdog Messages

$SubFuncStrucLen = 9 + (4 * ConnectionListCount)$

## See Also

Disable Broadcasts 0x2222 21 02 (page 648), Enable Broadcasts 0x2222 21 03 (page 649)

## Disable Broadcasts 0x2222 21 02

Notifies the server that a client does not want to receive messages from other clients.

**NetWare Server: 2.x, 3.x, 4.x, 5.x**

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (2)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

### Remarks

After receiving this request, the server refuses to let other client log messages for forwarding to this client.

Any client can call **Disable Broadcasts**.

## See Also

## Enable Broadcasts 0x2222 21 03 (page 649)

# Enable Broadcasts 0x2222 21 03

Enables message reception after message reception has been disabled by calling **Disable Broadcasts**.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (3)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Disable Broadcasts 0x2222 21 02 (page 648)**

# Get Broadcast Message 0x2222 21 11

Returns a message sent by another client.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (11)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	MessageLen	byte
9	Message	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Bad Station Number

## Remarks

If no message has been sent, *MessageLen* will contain zero.

**Get Broadcast Message** allows station numbers greater than 256. It also returns unique error codes on message receive failures.

Any client can call **Get Broadcast Message**.

## See Also

**Send Broadcast Message 0x2222 21 10 (page 654)**

# Get Broadcast Message (old) 0x2222 21 01

Returns a message sent by another client.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (1)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	MessageLen	byte
9	Message	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Bad Station Number

## Remarks

If no message has been sent, *MessageLen* will contain zero.

Any client can call **Get Broadcast Message**.

## See Also

**Send Broadcast Message (old) 0x2222 21 00 (page 656)**

# Log Network Message 0x2222 23 13

Places a message in a Novell file server's workstation log file.

**NetWare Server:** 2.x, 3.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2 + MessageLen)	word (Hi-Lo)
9	SubFunctionCode (13)	byte
10	MessageLen	byte
11	Message	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
136	0x88	Invalid File Handle
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error



Decimal	Hex	Description
255	0xFF	Failure, No Files Found, Lock Error

## Remarks

**Log Network Message** is a form of communication that is sometimes used for accounting purposes. The message is placed within the file server's NET\$LOG.MSG log file in the format:

```
mm/dd/yy hh:mm STN dd: Message
```

Message is the text received in the actual message. The date, time, and station number are added by the server.

<CR><LF> characters are added to the end of each line in the log file.

If NET\$LOG.MSG does not already exist, the server creates this file.

# Send Broadcast Message 0x2222 21 10

Sends a message to another client.

**NetWare Server:** 3.12, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (10)	byte
10	ClientListCount	word
12	ClientList	long[ClientListCount]
12 + ClientListCount * 4	MessageLen	byte
13 + ClientListCount * 4	Message	byte[MessageLen]

## Reply Format

Offset	Content	Type
Reply header		
8	RClientListCount	word
10	RClientCompFlag	long[ClientListCount]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Send Broadcast Message** allows station numbers greater than 256. It also returns unique error codes on message receive failures.

*RClientCompFlag* can have the following values:

- 0 Successful
- 1 Illegal Station Number

- 2 Client Not Logged In
- 3 Client Not Accepting Messages
- 4 Client Already Has a Message
- 150 No Alloc Space for the Message

*SubFuncStrucLen* = 4 + (4\* *ConnectionListCount*) + *MessageLen*

## See Also

**Get Broadcast Message 0x2222 21 11 (page 650)**

# Send Broadcast Message (old) 0x2222 21 00

Sends short text messages to a list of clients.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (21)	byte
7	SubFuncStrucLen	word
9	SubFunctionCode (0)	byte
10	ClientListLen	byte
11	TargetClientList	long[ClientListLen]
11 + ClientListLen	MessageLen	byte
12 + ClientListLen	Message	byte[MessageLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ClientListLen	byte
9	SendStatus	byte[ClientListLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Bad Station Number
255	0xFF	Failure

## Remarks

The message can be no longer than 58 bytes; longer messages will be truncated.

*SendStatus* indicates the result of trying to log the message for delivery to the target clients and can have the following values:

0x00 Success; the message has been recorded for later delivery

0xFF Failure; the target client does not exist or has signaled the server not to accept messages

0xFC Message Pending; the server is already holding a message for delivery to the target client and cannot accept another message

*SubFuncStrucLen* = 3 + *ClientListLen* + *MessageLen*

Any client can call **Send Broadcast Message (old)**.

## See Also

**Get Broadcast Message (old) 0x2222 21 01 (page 651), Get Broadcast Message 0x2222 21 11 (page 650), Send Broadcast Message 0x2222 21 10 (page 654)**





## NCP Extension

NCP Extension NCPs allows server application developers to write NLMs to be implemented in the NetWare OS as NCPs.

NetWare allows the NLM to create an NCP extension that calls a service request within the NLM. NetWare assigns an NCP number to the NCP extension.

A workstation application can send a custom request to the NLM over the existing NCP connection (which was established when the workstation attached to the file server) by retrieving the NCP number and calling **Execute NCP Extension**.

Applications distributed among servers and applications loaded on workstations benefit from NCP Extension primarily because of how simple it makes client-server communication. For example, workstation clients who use the NCP Extension can take advantage of established NCP client-server connections without being concerned with or even knowing about IPX/SPX. Opening sockets, preparing ECBs, filling out the IPX/SPX packet header, etc. is all managed by NCP Extension, which encourages application developers because the functionality provided by their NLMs are more accessible and true distributed applications are that much simpler.

To access NCP Extension NCPs, use the following:

- ♦ [Chapter 27, “Concepts,” on page 661](#)
- ♦ [Chapter 28, “NCPs,” on page 663](#)





# 27 Concepts

This section explains ideas that are common to NCP Extension NCPs.

## NCP Extension 37

Generally, NetWare's NCP Extensions are implemented as follows:

1. You write an NLM to perform some task within the server, such as accessing and retrieving information from a server database.
2. The NLM registers itself with the OS as NCP function number 37.

The NLM also identifies itself to the OS by passing a name such as "MyNCP."

3. The OS assigns and returns a subfunction number for the NLM.

The NCP is now a custom-built NCP Extension. Its function number is 37, and its subfunction number is assigned by the OS and maintained in a dynamic linked list in the OS until the NLM deregisters itself.

If the NLM unloads, deregistering itself from the OS, it goes away and its subfunction number is taken out of the linked list in the OS. When the NLM registers itself as an NCP again, it is assigned another subfunction number.

## NCP Extension 36

To allow for the dynamic nature of the NCP subfunction number, the OS also provides NCP 36, which allows workstation applications to call an NCP extension without knowing its subfunction number. The OS provides workstation applications the ability to retrieve an NCP Extension subfunction number as well as other NCP Extension information from the OS.

For example, a workstation application can call NCP 36, passing the name of an application NLM who has registered itself as an NCP. NCP 36 performs the following tasks as requested by the application:

- ♦ Scan currently loaded NCP extensions by number
- ♦ Scan currently loaded NCP extensions by name
- ♦ Returns the maximum NCP data size
- ♦ Returns an NCP subfunction number that corresponds to the name of a custom-built NCP such as "MyNCP"



# 28 **NCPs**

This section describes each of the NCP Extension NCPs, their Request and Reply formats, and Return Values.

# Execute NCP Extension 0x2222 37

Executes an NCP Extension that has been registered with the OS.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (37)	byte
7	SubFuncStrucLen (4 + sizeof RequestData)	word (Hi-Lo)
9	NCPextensionNumber	long (Hi-Lo)
13	RequestData	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	ReplyBuffer	byte[]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## Remarks

You must specify the NCP extension number that was obtained by calling **Get NCP Extension Information** (0x2222 36). *NCPextensionNumber* is equivalent to the NCP subfunction number.

Boundary checking is performed by the NCP extension handler.

The *RequestData* portion of the request packet is depending on the NCP extension handler that is used. If you use the extension handling capabilities of CLIB.NLM, the *RequestData* buffer might be as follows:

Offset	Content	Type
13	MaximumReplyLength	word (Lo-Hi)

Offset	Content	Type
15	RequestBuffer	byte[] /*528 bytes*/
	RequestBufferLength	word (Lo-Hi)

The reply buffer length varies depending on the protocol that is used. IPX will allow a reply buffer of only 536 bytes. IP will allow a bigger buffer depending on what packet size was negotiated on connection.

If you use the extension handling capabilities of CLIB.NLM, the reply buffer might be as follows:

Offset	Content	Type
8	ReplyBufferLength	word (Lo-Hi)
10	ReplyBuffer	byte[]

## See Also

**Get NCP Extension Information (old) 0x2222 36 00 (page 666), Get NCP Extension Maximum Data Size 0x2222 36 01 (page 668), Get NCP Extension Information by Name 0x2222 36 02 (page 669), Get Number of Registered NCP Extensions 0x2222 36 03 (page 671), Get NCP Extension Registered Verbs List 0x2222 36 04 (page 672), Return NCP Extension Information 0x2222 36 05 (page 673), Return NCP Extension Maximum Data Size 0x2222 36 06 (page 675)**

# Get NCP Extension Information (old) 0x2222 36 00

Returns information about the specified NCP Extension.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (0)	byte
10	NCPExtensionNumber	long (Lo-Hi)

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NCPextensionNumber	long (Lo-Hi)
12	NCPextensionMajorVersion	byte
13	NCPextensionMinorVersion	byte
14	NCPextensionRevisionNumber	byte
15	NCPextensionNameLength	byte
16	NCPextensionName	byte[]
48	NCPextensionCustomData	byte[]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## Remarks

If the value specified by *NCPextensionNumber* is not found, the next increasing NCP Extension number will be returned (if present).

It is recommended that you set *NCPExtensionNumber* to -1 initially. **Get NCP Extension Information (old)** scans for an NCP extension until the first valid NCP extension is found; information about that NCP extension is then returned. By using the *NCPextensionNumber* that is returned in the reply packet, you can make additional requests using this NCP until the desired NCP extension is found and the desired information is returned. (**Return NCP Extension Information 0x2222 36 05** looks only for a specified NCP extension and does not scan sequentially through loaded NCP extensions.)

*NCPextensionCustomData* is a 32-byte buffer of custom information that the NCP extension handler can optionally use.

## See Also

**Return NCP Extension Information 0x2222 36 05 (page 673), Execute NCP Extension 0x2222 37 (page 664)**

# Get NCP Extension Maximum Data Size 0x2222 36 01

Returns the maximum amount of data that may be sent in the packet.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (1)	byte

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NCPdataSize	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## See Also

**Return NCP Extension Maximum Data Size 0x2222 36 06 (page 675), Execute NCP Extension 0x2222 37 (page 664)**



# Get NCP Extension Information by Name 0x2222 36 02

Returns information about the NCP Extension specified by name.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (3 + NCPextensionNameLen)	word (Hi-Lo)
9	SubFuncCode (2)	byte
10	NCPextensionNameLen	byte
11	NCPextensionName	byte[NCPextensionNameLen]

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NCPextensionNumber	long (Lo-Hi)
12	NCPextensionMajorVersion	byte
13	NCPextensionMinorVersion	byte
14	NCPextensionRevisionNumber	byte
15	NCPextensionNameLength	byte
16	NCPextensionName	byte[NCPextensionNameLen]
48	NCPextensionCustomData	byte[]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## Remarks

*NCPextensionCustomData* is a 32-byte buffer containing custom information that the NCP extension handler can optionally use.

## See Also

**Execute NCP Extension 0x2222 37 (page 664)**

# Get Number of Registered NCP Extensions 0x2222 36 03

Returns the current number of NCP Extensions that have been registered.

NetWare Server: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (3)	byte

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NumberOfNCPExtensions	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## See Also

[Execute NCP Extension 0x2222 37 \(page 664\)](#)

# Get NCP Extension Registered Verbs List 0x2222 36 04

Returns a list of registered NCP Extension numbers, starting at the number specified by *StartingNumber*.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (4)	byte
10	StartingNumber	long (Lo-Hi)

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	ReturnedListCount	long
12	nextStartingNumber	long
16	NCPEExtensionNumbers	long[ReturnedListCount]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## Remarks

*StartingNumber* should be set to zero for the initial request. *nextStartingNumber* will contain the next number to start with for all subsequent requests so that all registered NCP Extensions are eventually returned. The list is complete when *nextStartNumber* is zero.

## See Also

**Execute NCP Extension 0x2222 37 (page 664)**

# Return NCP Extension Information 0x2222 36 05

Returns information about the NCP Extension specified by the extension number.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (5)	byte
10	NCPextensionNumber	long (Lo-Hi)

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NCPextensionNumber	long (Lo-Hi)
12	NCPextensionMajorVersion	byte
13	NCPextensionMinorVersion	byte
14	NCPextensionRevisionNumber	byte
15	NCPextensionNameLength	byte
16	NCPextensionName	byte[NCPextensionNameLength]
48	NCPextensionCustomData	byte[]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## Remarks

Return NCP Extension Information will return information only if the NCP Extension number matches.

*NCPextensionCustomData* is a 32-byte buffer of custom information that the NCP extension handler can optionally use.

## See Also

**Execute NCP Extension 0x2222 37 (page 664)**

# Return NCP Extension Maximum Data Size 0x2222 36 06

Returns the largest size of data that may be placed in the packet for the specified NCP Extension.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (36)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (6)	byte

## Reply Format (InformationType = 0)

Offset	Content	Type
Reply header		
8	NCPDataSize	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	

## See Also

**Execute NCP Extension 0x2222 37 (page 664)**





# XI

## NDS

NDS NCPs enable clients to communicate with the server, send or receive fragments, and manage connection status information.

To access NDS NCPs, use the following:

- ♦ [Chapter 29, “NCPs,” on page 679](#)
- ♦ [Chapter 30, “Structures,” on page 691](#)
- ♦ [Chapter 31, “NDS Attribute Syntaxes,” on page 693](#)



# 29 **NCPs**

This section describes each of the NDS NCPs, their Request and Reply formats, and Return Values.

# Clear Statistics 0x2222 104 07

Clears the statistics that can be reported by **Return Statistics**.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (7)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	BAD_PACKET

## See Also

**Return NDS Statistics 0x2222 104 06 (page 686)**

# Fragment Close 0x2222 104 03

Closes the FraggerHandle obtained by calling **Send NDS Fragmented Request/Reply** (0x2222 104 02).

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (7)	byte
8	FraggerHandle	LONG (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	FAILURE

## See Also

**Send NDS Fragmented Request/Reply 0x2222 104 02 (page 688)**

# Monitor NDS Connection 0x2222 104 05

NetWare Server: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (5)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	
254	0xFE	BAD_PACKET

# Ping for NDS NCP 0x2222 104 01

Queries the server.  
**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (1)	byte
8	Reserved (0)	byte[3]

## Reply Format

Offset	Content	Type
Reply header		
8	PingVersion	long (Lo-Hi)
12	TreeName	unicode
xx	RootMostEntryDepth	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	NDS NCP not available
254	0xFE	BAD_PACKET

## Remarks

You'll get back the ping version number, the tree name, the NDS version, and the distance from the root of the root most entry on the server.

# Reload NDS Software 0x2222 104 08

Reloads and restarts the NDS software.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (8)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Status	Int4
12	Flags (0)	Int4

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	BAD_PACKET

## Remarks

A supervisor with sufficient privileges must call **Reload NDS Software**.

There is no interruption of service when **Reload NDS Software** is called.

*Status* is zero for success; otherwise, it is nonzero.



# Return Bindery Context 0x2222 104 04

Returns the bindery context.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (4)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	length	long
12	binderyContext	unicode (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	BAD_PACKET

## Remarks

The Unicode type of *binderyContext* means that each character in the string of characters that compose this parameter are each 16 bits in length. The first bit holds the character itself; the second bit holds the NULL terminator. For example, if the bindery context were as follows:

O=Novell

the content of binderyContext would be as follows:

O=NOVELL

Each character is NULL terminated, and the string as a whole is also NULL terminated. The NULL terminator for the whole string is 16 bits; therefore, each byte of the last 16 bits is filled with NULL.

# Return NDS Statistics 0x2222 104 06

Returns several statistics about NDS operation.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (6)	byte
8	RequestFlags	Int4

## Reply Format

Offset	Content	Type
Reply header		
8	Statistics (variable)	Int4

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	BAD_PACKET

## Remarks

*RequestFlags* can have the following values:

0x00000001 Output Fields  
0x00000002 No Such Entry  
0x00000004 Local Entry  
0x00000008 Type Referral  
0x00000010 Alias Referral  
0x00000020 Request Count  
0x00000040 Request Data Size  
0x00000080 Reply Data Size  
0x00000100 Transport Referral  
0x00000200 Transport Referral

0x00000400 Up Referral  
0x00000800 Down Referral

Each *RequestFlag* value that is set to one requests a particular statistic. Each statistic is an Int4. The reply indicates in the Output Fields statistic which *RequestFlags* are being returned. The items that are both requested and supported appear contiguously in the reply.

The request is invalid if both 1) the *RequestFlags* do not include a request for the Output Fields and 2) the responding server does not support a requested statistic.

*RequestFlags* and statistics can be added in the future.

## See Also

**Clear Statistics 0x2222 104 07 (page 680)**

# Send NDS Fragmented Request/Reply 0x2222 104 02

Sends fragmented requests to a server and returns fragmented replies.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (104)	byte
7	SubFunctionCode (2)	byte
8	FraggerHandle	long (Lo-Hi)
12	FragSize	long (Lo-Hi)
16	TotalRequest	long (Lo-Hi)
20	Flags	long (Lo-Hi)
24	Verb	long (Lo-Hi)
28	ReplyBufferSize	long (Lo-Hi)
32	RequestData	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ReplySize	long (Lo-Hi)
12	FraggerHandle	long (Lo-Hi)
16	ReplyData	byte[]

## Parameters

*FraggerHandle*

(Request) Is initially -1.

*FragSize*

(Request) Specifies the number of data bytes that the client is expecting.

*TotalRequest*

(Request) Specifies the total number of bytes in the message.

### *Flags*

(Request) Is always set to zero.

### *Verb*

(Request) Specifies the number of the NDS verb that is to be executed.

### *ReplyBufferSize*

(Request) Specifies the number of bytes in the reply buffer.

### *ReplySize*

(Reply) Specifies the size of the reply packet.

### *FraggerHandle*

(Reply) Specifies the unique identifier assigned by the server.

### *ReplyData*

(Reply) Specifies the data returned, which is dependent upon the chosen NDS verb.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	SERVER_OUT_OF_MEMORY
226	0xE2	TOO_FEW_FRAGMENTS
227	0xE3	TOO_MANY_FRAGMENTS
228	0xE4	PROTOCOL_VIOLATION
229	0xE5	SIZE_LIMIT_EXCEEDED
251	0xFB	UNKNOWN_REQUEST
253	0xFD	INVALID_PACKET_LENGTH
254	0xFE	BAD_PACKET
255	0xFF	FAILURE

## Remarks

The reply *FraggerHandle* can change with each fragment received, so make sure this number is copied to the next request header. When *FraggerHandle* is -1L, there is no more data to receive.

## See Also

**Clear Statistics 0x2222 104 07 (page 680)**



# 30 Structures

This section describes the NDS structures and their fields.

# Attribute

## Syntax

```
struct Attribute
{
    unicode    AttrName;
    any_t      AttrValue[];
};
```

## Remarks

The AttrValue type indicates that an attribute value is encoded, based on the syntax of the attribute, in this field. For example, the syntax of the Back Link attribute is as follows:

```
int32    length;
int32    remoteID;
unicode  serverName;
```

Each syntax (see “**NDS Attribute Syntaxes**” on page 693) has its own encoding. Therefore, any\_t denotes one of the defined encoded syntaxes. For example, if you filled in the Attribute structure with the Back Link attribute, you would produce the following:

```
struct Attribute
{
    unicode    BackLink; //attribute name
    int32      length;    //syntax of attribute
    int32      remoteID;
    unicode    serverName;
};
```



# 31

## NDS Attribute Syntaxes

This section describes each of the NDS Attribute Syntaxes and their encoding rules used to pass an attribute over the network.

## Back Link

Internally manages the directory.

### Syntax

```
int32    length;  
int32    remoteID;  
tdr_str   serverName;
```

# Boolean

Represents true or false for an attribute's value.

## Syntax

```
int32  length;  
int8   boolean;
```

## Remarks

To the directory, one represents true and zero represents false.

# Case Exact String

Represents an attribute's value as a case-sensitive string.

## Syntax

```
tdr_str    caseExactString;
```

## Remarks

Case Exact String is used when performing case-sensitive comparisons.

Two Case Exact Strings match for equality when they have the same length and their corresponding characters are identical. Leading spaces, trailing spaces, and multiple consecutive internal spaces are not significant. (Multiple consecutive internal spaces are treated as a single space character.) The directory might omit insignificant spaces when storing an attribute's value.

# Case Ignore List

Represents an attribute's value as a sequence of case ignore strings.

## Syntax

```
tdr_str    caseIgnoreStringList[];
```

## Remarks

Two Case Ignore Lists match for equality if and only if the number of case ignore strings in each list is the same and their corresponding strings match according to the rules of equality for Case Ignore Strings.

# Case Ignore Strings

Represents an attribute's value as a case-insensitive string.

## Syntax

```
tdr_str    caseIgnoreString;
```

## Remarks

Case Ignore String is used when performing case-insensitive comparisons.

Two Case Ignore Strings match for equality when they have the same length and their corresponding characters are identical in all respects except for case. Leading spaces, trailing spaces, and multiple consecutive internal spaces are not significant. (Multiple consecutive internal spaces are treated as a single space character.) The directory might omit insignificant spaces when storing an attribute's value.

## Class Name

Represents an attribute's value as an object class name.

### Syntax

```
tdr_str  className;
```

### Remarks

The matching rules for Class Name are the same as for Case Ignore String.

# Counter

Represents an attribute that is used to count.

## Syntax

```
int32  length;  
int32  integer;
```

## Remarks

As an example, Counter can be used to keep track of the number of grace logins allowed.

Counter attributes are single valued. Counter is different from the Integer attribute because Counter values increase or decrease the total of that attribute (rather than simply zeroing out the old value and changing it to the new value).



## Distinguished Name

Represents an attribute's value as the name of an object in the NDS tree.

### Syntax

```
tdr_str    distinguishedName;
```

## Email Address

Represents an attribute's value as an email address on the network.

### Syntax

```
int8    octetString;
```

# Facsimile Telephone Number

Represents an attribute's value as a string that complies with the internationally designated format for telephone numbers, E.123, and an optional bit string formatted according to Recommendation T.30.

## Syntax

```
int32      length;      //total length
tldr_str   telephoneNumber;
int32      bitCount;
int32      parameterLen[];
int8       parameters[];
```

## Remarks

The directory matches Facsimile (Fax) Telephone Numbers based on the telephone number field. The rules for matching fax telephone numbers are identical to those for the Case Exact String, except that the directory skips all space and hyphen characters during the comparison.

The number of parameter bytes for *bitCount* is determined by dividing *bitCount* by 8. The number of bytes for *parameters* is determined using the same formula that *bitCount* uses.

# Hold

Represents an attribute's value as an object name/level amount.

## Syntax

```
int32      length;  
int32      amount;  
tdr_str    dn;
```

# Integer

Represents an attribute's value as an integer.

## Syntax

```
int32    length;  
int32    integer;
```

## Remarks

If you need a counter; use the Counter syntax.

Two Integer attribute values match for equality if they are the same. The ordering rules for integers apply to attribute values.

# Interval

Represents an attribute's value as an interval of time.

## Syntax

```
int32    length;  
int32    integer;
```

# Net Address

Represents a network address in a NetWare environment.

## Syntax

```
int32    length;    //length of address + 4
int32    addressType;
int8     address[];
```

## Remarks

Two Net Address attribute values match for equality if the type, length, and value of the network address match.

The address length is expressed in bytes. Since the address itself is stored as a binary string, each 4-bit nibble must be converted to the correct hexadecimal character before the address can be displayed as a hexadecimal address.

# Numeric String

Represents an attribute's value as a numeric string as defined in CCITT X.208.

## Syntax

```
tdr_str    numericString;
```

## Remarks

Two Numeric String attribute values match for equality if the strings are the same length and their corresponding characters are identical.

The numeric string character set consists of the digits 0 through 9 and the space character.

During comparisons of numeric strings, the directory skips all spaces. Since spaces are insignificant for comparisons, the directory might omit them when storing a Numeric String attribute value.



# Object ACL

Represents an attribute's value as an access control list entry.

## Syntax

```
int32      length;      //total length of ACL entry
tdr_str    protectedAttrName;
tdr_str    subjectName;
```

## Remarks

An object ACL value can protect either an object or an attribute. The protected object is always the one that contains the ACL attribute.

Object ACL supports both matching for equality and approximate matching, with the difference concerning the privileges field of the comparison value.

When matching for equality, the privilege set must match exactly for the comparison to succeed.

When matching approximately, any bits in the privilege field in the filter that are set must also be set in the target. The comparison ignores other bits in the target.

Values with the same *protectedAttrName* and *subjectName* are considered to be duplicate values and are not permitted.

## Octet List

Represents an attribute's value as an ordered sequence of octet strings.

### Syntax

```
octetString    octetStringList[];
```

### Remarks

A presented octet list matches a stored list if the presented list is a subset of the stored list.

# Octet String

Represents an attribute's value as an octet string.

## Syntax

```
int8    octetString[];
```

## Remarks

Two Octet Strings match if they are the same length and their corresponding bit sequences (octets) are identical. The comparison operation uses the first pair of octets that do not match to determine the order of the strings.

# Path

Represents an attribute's value as a file system path.

## Syntax

```
int32      totalLength;  
int32      nameSpaceType;  
tdr_str    volumeDN;  
int8       address[];
```

## Remarks

The directory compares the string represented by address using the same rules as those for the Case Exact String syntax.

# Postal Address

Represents an attribute's value as a postal address.

## Syntax

```
tdr_str    caseIgnoreStringList[];
```

## Remarks

An attribute value for a Postal Address typically includes selected attributes from the MSH Unformatted Postal O/R Address version 1 according to Recommendation f.401.

The value is limited to six lines of 30 characters each, including a Postal Country name. Usually, the information contained in an address includes the following:

- ♦ Addressee's name
- ♦ Street address
- ♦ City, state, or province
- ♦ Postal code
- ♦ Postal office box number

The items included in the address depend on the specific requirements for the named object.

The matching rules for Postal Address are the same as those for Case Ignore List.

The count of the string must equal 6.

# Printable String

Represents an attribute's value as a printable string as defined in CCITT X.208.

## Syntax

```
tdr_str    printableString;
```

## Remarks

The case (upper or lower) is significant when comparing Printable Strings.

Two Printable Strings match for equality when they have the same length and their corresponding characters are identical. Leading spaces, trailing spaces, and multiple consecutive internal spaces are not significant. (Multiple consecutive internal spaces are treated as a single space character.) The directory might omit insignificant spaces when storing an attribute value.

The printable string character set contains the following:

- ♦ A...Z uppercase alphabetic letters
- ♦ a...z lowercase alphabetic letters
- ♦ 0...9 digits
- ♦ space character
- ♦ ' apostrophe
- ♦ ( left parenthesis
- ♦ ) right parenthesis
- ♦ + plus sign
- ♦ , comma
- ♦ - hyphen
- ♦ . full stop (period)
- ♦ / solidus (slash)
- ♦ : colon
- ♦ = equal sign
- ♦ ? question mark

# Replica Pointer

Represents an attribute's value as a partition replica.

## Syntax

```
int32      length;
tdr_str    replicaServerName;
int32      replicaType;
int32      replicaNumber;
Referral   address[];

struct    Referral
{
    int32   type;
    int8    address[];
};
```

## Remarks

Each value is composed of four parts:

- ♦ The complete name of the name server that stores the replica
- ♦ A value describing the capabilities of this copy of the partition: master, secondary, read-only
- ♦ A number representing the replica
- ♦ A value indicating the replica number
- ♦ A network address giving a hint for an address where the name server probably resides

When the directory matches Replica Pointer values, it compares only the *replicaServerName* field.

*length* specifies the total length of the replica pointer in bytes.

## Stream

Represents an attribute whose values might exceed the maximum allowed value size.

## Syntax

```
int8    octetString[];
```

## Remarks

Any attribute defined with this syntax is single valued.



# Telephone Number

Represents an attribute's value as a telephone number.

## Syntax

```
tdr_str    telephoneNumberString;
```

## Remarks

The rules for matching Telephone Number syntaxes are identical to those for the Case Exact String, except that all space and hyphen characters are skipped during the comparison.

# Time

Represents an attribute's value as a time.

## Syntax

```
int32    length;  
int32    integer;
```

## Remarks

Time consists of a whole number of seconds, where zero equals 12:00 midnight (a.m.), January 1, 1970, GMT.

The directory compares two Time values by comparing their integer values.

# Timestamp

Represents an attribute's value as a time when a particular event occurred or will occur.

## Syntax

```
int32    length;    //length = 8
int32    seconds;
int16    replicaNumber;
int16    event;
```

## Remarks

Timestamp consists of a whole number of seconds, where zero equals 12:00 midnight (a.m.), January 1, 1970, GMT.

The event is an integer that further orders events occurring within the same whole-second interval.

The directory compares two Timestamp values by comparing the second fields first and then the event fields. If the seconds are unequal, the directory determines order by that field only.

The order of comparisons follows:

1. seconds
2. replica number
3. event

The directory compares until one of these fields is not equal.

# TypedName

Represents an attribute's value as a level and an interval associated with an object.

## Syntax

```
int32      length;  
int32      level;  
int32      interval;  
tdr_str    dn;
```

## Remarks

The *level* of the attribute indicates its priority.

The *interval* indicates the frequency of reference.

The *level* and *interval* values are user assigned and interpreted. To be effective, they must be implemented by the user, who can use them to implement iterative intervals or enforce priority.

# Unknown

Represents an attribute whose attribute definition was deleted from the schema.

## Syntax

```
int8    octetString[];
```

## Remarks

An Unknown syntax cannot be matched.



# XII

## Packet Burst

To access Packet Burst NCPs, use the following:

- ♦ [Chapter 32, “NCPs,” on page 725](#)





# 32 NCPs

This section describes each of the Packet Burst NCPs, their Request and Reply formats, and Return Values.

The following Packet Burst NCPs are not documented yet, but will be available in future releases:

- ♦ 0x2222 99
- ♦ 0x2222 100
- ♦ 0x2222 102
- ♦ 0x2222 103

# Packet Burst Connection Request 0x2222 101

Connects a packet burst.

## Syntax

Offset	Content	Type
Request header		
6	FunctionCode (101)	byte
7	LocalConnectionID (random)	long (Hi-Lo)
11	LocalMaxPacketSize	long (Hi-Lo)
15	LocalTargetSocket (from IPXOpenSocket)	word (Hi-Lo)
17	LocalMaxSendSize	long (Hi-Lo)
21	LocalMaxRecvSize	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
6	CompletionCode (0)	byte
7	RemoteTargetID (random)	long (Hi-Lo)
11	RemoteMaxPacketSize	long (Hi-Lo)

## Parameters

### *LocalConnectionID*

(Request) Specifies the randomly generated number used by the shell to reference its connection with a server.

### *LocalMaxPacketSize*

(Request) Specifies the desired maximum packet size.

### *LocalTargetSocket*

(Request) Specifies the IPX socket being used at the workstation to send and receive burst packets.

### *LocalMaxSendSize*

(Request) Is reserved for future use.

*LocalMaxRecvSize*

(Request) Is reserved for future use.

*RemoteTargetID*

(Reply) Specifies the identification number used by the server in connection with the shell (similar to connection numbers used with SPX).

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If there is enough memory in a workstation to handle packet bursts, the shell will explicitly attempt to make a packet burst connection. This process is automatically performed by the shell for each NetWare server connection.

The packet size for burst packets is negotiated in the same manner as for non-packet burst packet sizes.

Once the packet burst connection is made, is it automatically implemented whenever a read or write request involves more than 512 bytes of data. Multiple packets or fragments make up a burst. The burst header (36 bytes) follows the IPX header (30 bytes), which is followed by the fragment data or the missing fragments list.

Offset	Length	Field Name	Description
0	2	Packet Type	The packet purpose identifier: 0x1111 Allocate Slot Request 0x2222 Server NCP Request 0x3333 Server NCP Reply 0x5555 Deallocate Slot Request 0x7777 Packet Burst Packet 0x9999 Positive Acknowledgement
2	1	Stream Type	The type of burst. Only the server uses this field. 0x02 is the only valid type.
3	1	System Flags	0x04 (ABT) Set for an abort request. The SYS bit is set if ABT is set.  0x10 (EOB) Set for an end of burst. It will be set in the last packet of a burst.  0x80 (SYS) Set for a system packet. No burst data accompanies or follows these packets.
4	4	Source Connection ID	The workstation's connection identification number.

Offset	Length	Field Name	Description
8	4	Destination Connection ID	The server's connection identification number.
12	4	Packet Sequence Number	(Hi-Lo) Incremented for each packet sent by a node. It represents the sequence number of this packet within a burst.
16	4	Send Delay Time	(Hi-Lo) The delay time between consecutive packet sends in 100 microsecond increments.
20	2	Burst Sequence #	(Hi-Lo) The sequence number of each packet within a burst, beginning with zero.
22	2	ACK Sequence #	(Hi-Lo) The burst sequence number that is expected next.
24	4	Burst Length	(Hi-Lo) The total length of the data to be received in the packets of this burst.
28	4	Data Offset	(Hi-Lo) The offset into the total burst data at which this packet's data should be located.
32	2	Data Bytes	(Hi-Lo) The number of data bytes in this packet.
34	2	Missing Fragment List Count	(Hi-Lo) The number of missing fragments that are being reported in this packet. The list immediately follows as part of this packet if this value is greater than zero. The system bit is set if this field is used.

A system packet (indicated by the SYS bit in the burst header) can be used to report fragments of a burst that were missed. For each missed fragment, the following information will be reported following the burst header:

(Hi-Lo) 4-byte value specifying the offset into the burst where the missing data begins.

(Hi-Lo) 2-byte value specifying the number of bytes of missing data.

If there is more than one "hole" of missing data in the burst, there will be a corresponding number of missing fragments reported in the list.

Prior to sending bursts of data, a burst request must be issued. There are two types of burst requests: one to read and one to write. The request is identified by data following the burst header information as follows:

Offset	Length	Field Name	Description
0	4	Function Number	(Lo-Hi) 1 = read; 2 = write
4	4	File Handle	Supply this handle as it is without reformatting it.
8	8	Reserved	All zeros.
16	4	File Offset	(Hi-Lo) The starting offset in the file.
20	4	Number Of Bytes	(Hi-Lo) The number of bytes to be written or read.
(24)	??	Data	The first packet of data to be written if this is a function 2 write request.

As read requests are processed, the first packet of each burst contains:

- ♦ (Lo-Hi) 4-byte result code
- ♦ 4-byte value indicating the number of bytes read
- ♦ The first portion of the data that was read

The possible result codes for a read request are:

- 0 No error
- 1 Initial error
- 2 I/O error
- 3 No data read

Each burst of a write request is responded to with a burst packet that contains a 2-byte result code (Lo-Hi). One write request is issued, but each burst of the request is replied to.

The possible result codes for a write request are:

- 0 Write successful
- 4 Write error

### **NetWare 3.11 Notes**

The packet burst NLM must be loaded in order for **Packet Burst Connection Request** to be available.

The local workstation (or client) determines the maximum packet size by calling the IPX 1Ah function. The local maximum burst size is determined by the following formula:

$$P * (S + 102)$$

where P is the number of packets requested in NET.CFG using the PB BUFFERS = <n> statement, and S is the negotiated packet size. If P is zero, packet burst will not be used. If P is 1, it will be incremented to 2. If P is greater than 10, the value will be changed to 10. If the workstation memory is insufficient to accommodate the requested buffers, P will be adjusted.



# XIII

## Print

Print NCPs enable clients to spool data to the file server to eventually be printed on one of the network printers.

Clients can retrieve a printer's current job queue and can delete any of their own jobs from that queue. Clients can also query the server for the current status of a network printer.

To access Print NCPs, use the following:

- ♦ [Chapter 33, "NCPs," on page 733](#)





# 33 NCPs

This section describes each of the Print NCPs, their Request and Reply formats, and Return Values.

# Close Spool File 0x2222 17 01

Closes the client's current spool file.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (1)	byte
10	AbortQueueFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
135	0x87	Create Filename Error
136	0x88	Invalid File Handle
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
147	0x93	No Read Privileges
148	0x94	No Write Privileges
149	0x95	File Detached
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle

Decimal	Hex	Description
156	0x9C	Invalid Path
157	0x9D	No Directory Handles
161	0xA1	Directory I/O Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
218	0xDA	Queue Halted
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
252	0xFC	No Such Object
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	Bad Printer, Lock Error, Hard Failure, No Files Found

## Remarks

If *AbortQueueFlag* is zero, the spool file is placed at the end of the print queue. If *AbortQueueFlag* is set, the spool file is not placed in the print queue; instead, the spool file's status flags are examined.

If the spool file is marked for deletion after printing, the file is immediately deleted; if not, the print server closes the spool file and disregards it.

## See Also

**Set Spool File Flags 0x2222 17 02 (page 741)**

# Create Spool File 0x2222 17 09

Creates and names a print spool and specifies its location.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (3 + FileNameLength)	word (Hi-Lo)
9	SubFunctionCode (9)	byte
10	DirectoryHandle	byte
11	FileNameLength	byte
12	FileName	byte[FileNameLength]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
132	0x84	No Create Privileges
135	0x87	Create Filename Error
141	0x8D	Some Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
148	0x94	No Write Privileges
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path

Decimal	Hex	Description
161	0xA1	Directory I/O Error
253	0xFD	Bad Station Number
254	0xFE	Directory Locked
255	0xFF	Bad Printer, Lock Error, Hard Failure, No Files Found

## Remarks

The client must have create and write privileges in the specified directory for **Create Spool File** to succeed.

**Create Spool File** is provided by the print server as a service to the client. If the client spools data to the print server without first opening a spool file, the print server automatically creates a spool file for the client in the print server's work directory. Because spool files created by the print server are not directly available to all clients, the print server allows a client to specify a location for the spool file, which allows the client to keep the spool file for future use after the print server has filled it with spooled data.

## See Also

**Write To Spool File 0x2222 17 00 (page 746)**

# Get Printer's Queue 0x2222 17 10

Returns the Object ID of the queue servicing the specified printer.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (10)	byte
10	PrinterNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ObjectID	long

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
255	0xFF	Bad Printer

## See Also

**Write To Spool File 0x2222 17 00 (page 746)**

# Get Printer Status 0x2222 17 06

Checks the status of a shared printer.

NetWare Server: 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (6)	byte
10	TargetPrinter	byte

## Reply Format

Offset	Content	Type
Reply header		
8	PrinterHalted	byte
9	PrinterOffLine	byte
10	CurrentFormType	byte
11	RedirectedPrinter	byte

## Parameters

### *PrinterHalted*

(Reply) Specifies whether the printer has been halted from the system console:

- 0xFF Printer is halted
- 0 Printer is not halted

### *PrinterOffLine*

(Reply) Specifies whether the printer is off-line.

### *CurrentFormType*

(Reply) Specifies the number of the form currently mounted on the printer.

### *RedirectedPrinter*

(Reply) Specifies the number of the printer that jobs destined for *TargetPrinter* are being sent to.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
251	0xFB	Bad Dir Handle
253	0xFD	Bad Station Number
255	0xFF	Bad Printer

## Remarks

**Get Printer Status** works only on a 286 server (NetWare 2.15 and above) when a printer is configured into the system with the NETGEN utility. The "P" command at the console must respond with a list of printers for this request to work.

If *RedirectedPrinter* and *TargetPrinter* are not the same, the target printer has been redirected from the system console, and jobs sent to it are automatically rerouted and placed in a print queue of the printer specified by *RedirectedPrinter*.



# Set Spool File Flags 0x2222 17 02

Checks the status of a shared printer.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (21)	word (Hi-Lo)
9	SubFunctionCode (2)	byte
10	PrintFlags	byte
11	TabSize	byte
12	TargetPrinter	byte
13	Copies	byte
14	FormType	byte
15	Reserved	byte
16	BannerName	byte[14]

## Parameters

### *PrintFlags*

(Request) Specifies the following bits:

- 3 Suppress form feeds
- 5 Delete the spool file after printing
- 6 Expand the tabs in the file
- 7 Print the banner page

### *TabSize*

(Request) Specifies tab stops should be set in every column that is a multiple of the value specified.

### *TargetPrinter*

(Request) Specifies which of the print server’s printers the data file should be queued to (printer numbers start at zero).

### *Copies*

(Request) Specifies the number of times the spool file should be sent to the printer.

### *FormType*

(Request) Specifies the form number the client wants the job to be printed on.

### *BannerName*

(Request) Specifies the filename the client wants to appear on the print banner page (NULL-padded if fewer than 14 characters).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Bad Printer

## Remarks

When a print server adds a job to the print queue, it records the client's current print parameters and later uses them to control printing the job.

*TabSize* has meaning only if bit 6 is set in *PrintFlags*. If bit 6 is set, the print server assumes that the file being printed is a standard ASCII file. Any tabs that are encountered while printing the file will be replaced by enough spaces to move the printer over to the next tab stop column. If tabs are being expended by the print server, the print server will interpret any Control Z character (0x1A) in the print file as an end-of-file mark and will stop printing the file.

If the tab expansion bit (bit 6) is cleared in *PrintFlags*, the printer makes no assumptions about the nature of the file it is printing or the nature of the device the file is being sent to. If bit 6 is cleared, the print server can drive many different types of devices (printer, plotters, etc.), but the burden of appropriately controlling these devices rests with the client using the device.

After each job is added to the queue, the print server resets the print parameters to the default values as follows:

- ♦ *PrintFlags* 0 (no banner, no tabs, no file delete)
- ♦ *TabSize* 8

- ♦ *TargetPrinter* 0
- ♦ *Copies* 1
- ♦ *FormType* 0
- ♦ *BannerName* 0 (no banner name)

## See Also

**Close Spool File 0x2222 17 01 (page 734), Write To Spool File 0x2222 17 00 (page 746), Get Printer's Queue 0x2222 17 10 (page 738)**

# Spool A Disk File 0x2222 17 03

Sends an existing file to a print queue so the client does not need to recopy the file into a separate spool file.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (3 + FileNameLength)	word (Hi-Lo)
9	SubFunctionCode (3)	byte
10	DirectoryHandle	byte
11	FileNameLength	byte
12	FileName	byte[FileNameLength]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
128	0x80	Lock Fail
129	0x81	Out Of Handles
135	0x87	Create Filename Error
136	0x88	Invalid File Handle
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
147	0x93	No Read Privileges
148	0x94	No Write Privileges
149	0x95	File Detached
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error

Decimal	Hex	Description
153	0x99	Directory Full Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
157	0x9D	No Directory Handles
161	0xA1	Directory I/O Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
218	0xDA	Queue Halted
232	0xE8	Write To Group
234	0xEA	No Such Member
235	0xEB	Property Not Set Property
236	0xEC	No Such Set
252	0xFC	No Such Object
254	0xFE	Directory Locked
255	0xFF	Bad Printer, Lock Error, Hard Failure, No Files Found

## Remarks

The server uses the client's current print parameters and adds the file to the job queue of the appropriate printer.

Clients must have open and read privileges for any files they spool.

# Write To Spool File 0x2222 17 00

Appends the specified data to the end of the client's current spool file.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (17)	byte
7	SubFunctionStrucLen (2 + DataLength)	word (Hi-Lo)
9	SubFunctionCode (0)	byte
10	DataLength	byte
11	Data	byte[DataLength]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Out Of Disk Space
128	0x80	Lock Fail
129	0x81	Out Of Handles
135	0x87	Create Filename Error
136	0x88	Invalid File Handle
141	0x8D	Some Files In Use
142	0x8E	All Files In Use
143	0x8F	Some Read Only
144	0x90	All Read Only
148	0x94	No Write Privileges
149	0x95	File Detached
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
153	0x99	Directory Full Error

Decimal	Hex	Description
161	0xA1	Directory I/O Error
162	0xA2	I/O Lock Error
255	0xFF	Failure, No Files Found

## Remarks

If the client does not have a current spool file, the print server creates the file in its work area and writes the data to it.

The print server makes no assumptions about the format or content that it writes to a spool file; the client must create a data stream that produces the desired output when it is sent to the target printer.

## See Also

**Close Spool File 0x2222 17 01 (page 734), Create Spool File 0x2222 17 09 (page 736), Get Printer's Queue 0x2222 17 10 (page 738)**





# XIV

## Queue

A queue is a bindery object with the item property, Q\_DIRECTORY, which has a 128-byte, NULL-terminated segment containing the directory path name of a directory where queued files can be created and maintained. Q\_DIRECTORY normally contains the directory path SYS:\SYSTEM\QUEUE\_ID\_#; QUEUE\_ID\_# is the unique object ID number by which the queue is known to the bindery.

A queue consists of the following:

- ♦ An object name
- ♦ An object type
- ♦ An object security level (usually 0x30)
- ♦ The item property, Q\_DIRECTORY, that contains the associated directory path of the queue directory queue job files are located
- ♦ The set property, Q\_SERVERS, that contains a list of all servers that are authorized to service jobs in (or remove jobs from) the queue (security of 0x30)
- ♦ The set property, Q\_OPERATORS, that contains a list of all objects (users) authorized to manipulate the queue (security of 0x32)

A queue can also have the set property, Q\_USERS, that contains a list of all user objects that are authorized to use the queue. If this property does not exist, any object can use the queue. The property security should be 0x30.

To access Queue NCPs, use the following:

- ♦ [Chapter 34, “Concepts,” on page 751](#)
- ♦ [Chapter 35, “NCPs,” on page 753](#)
- ♦ [Chapter 36, “Structures,” on page 825](#)



# 34 Concepts

This section explains ideas that are common to Queue NCPs.

## Status Flags

Possible status flags include:

- 1 (0x01) Queue Server Cannot Add Jobs
- 2 (0x02) Queue Servers Cannot Attach
- 4 (0x04) Queue Server Cannot Service Job

## Return Values

Possible return values include the following:

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure, File Exists Error



# 35 NCPs

This section describes each of the Queue NCPs, their Request and Reply formats, and Return Values.

# Abort Servicing Queue Job 0x2222 23 132

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (132)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Close File And Start Queue Job 0x2222 23 127 (page 768), Finish Servicing Queue Job 0x2222 23 131 (page 782), Create Queue Job And File 0x2222 23 121 (page 773)**

# Abort Servicing Queue Job (old) 0x2222 23 115

Closes a job file and resets the calling server's access rights to the file server to their original login values.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (115)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure, File Exists Error

## Remarks

Only a queue server that has previously accepted a job for service can call **Abort Servicing Queue Job (old)**.

**Abort Servicing Queue Job (old)** allows a server to inform the queue manager that it cannot complete servicing a job that was previously accepted for service.

**Abort Servicing Queue Job (old)** also checks a job's service restart flag to ensure that the job can be restarted automatically after a server failure. If this flag is set, the job's server station, server task, and server ID number fields are cleared; and the job remains in its current position in the queue. If the service restart flag is zero, the job entry is destroyed and the job file is deleted.

An aborted job returns to its former position in the job queue if its service restart flag is set. For example, if a job was at the beginning of the queue before being called, it would return to the beginning of the queue after being aborted. An aborted job could then be next in line for service. A server should not abort a job because of an error in the job's format or requests; the server should call **Finish Servicing A Queue Job 0x2222 23 131** to remove the job from the queue.

A server should abort a job only if some temporary internal problem prevents it from completing that job. For example, a print server might abort a job if the paper in the printer is jammed. When the print server calls the same job after the paper jam is fixed, it should be able to service the job successfully. A server should not abort a job that is attempting to access data without proper security clearance. If aborted, the job would remain in the queue and be serviced and aborted repeatedly.

## See Also

**Close File And Start Queue Job (old) 0x2222 23 105 (page 769), Finish Servicing Queue Job (old) 0x2222 23 114 (page 783), Create Queue Job And File (old) 0x2222 23 104 (page 775)**



# Attach Queue Server To Queue 0x2222 23 111

Attaches a station to the specified queue as a queue server so that the station can service jobs from the queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (111)	byte
10	QueueID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
234	0xEA	No Such Number
252	0xFC	No Such Object

Decimal	Hex	Description
255	0xFF	Failure, File Exists Error

## Remarks

You must be security equivalent to one of the objects listed in the queue's Q\_SERVERS group property to call **Attach Queue Server To Queue**.

A queue can have as many as 25 servers attached to it at a time. If 25 servers are already attached to the target queue, Maximum Queue Servers is returned.

## See Also

**Detach Queue Server From Queue 0x2222 23 112 (page 780)**

# Change Job Priority 0x2222 23 130

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFunction (130)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)
18	Priority	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Change Queue Job Entry 0x2222 23 123 (page 760)**

# Change Queue Job Entry 0x2222 23 123

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + JobStructure)	word (Hi-Lo)
9	SubFunction (123)	byte
10	QueueID	long (Hi-Lo)
14	<a href="#">JobStruct (page 826)</a>	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

[Change Job Priority 0x2222 23 130 \(page 759\)](#), [Get Queue Job List 0x2222 23 129 \(page 788\)](#), [Get Queue Job File Size 0x2222 23 135 \(page 785\)](#)

# Change Queue Job Entry (old) 0x2222 23 109

Changes information for a queue job entry.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + JobStruct)	word (Hi-Lo)
9	SubFunctionCode (109)	byte
10	QueueID	long (Hi-Lo)
14	JobStruct (page 826)	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure, File Exists Error

## Remarks

The client that created the job or an operator can call **Change Queue Job Entry (old)**.

The client selects the queue entry to be changed by designating the job number in JobStruct. The client can replace the information in the following fields:

TargetServerIDNumber

TargetExecutionTime

JobType

UserHoldFlag

ServiceRestartFlag

ServiceAutoStartFlag

TextJobDescription

ClientRecordArea

OperatorHoldFlag (if the calling client is an operator)

You can call **Change Queue Job Entry (old)** with **Read A Job Queue's Entry** to change a portion of the job record. However, if the target entry is already being servicing, Queue Servicing is returned and no changes are made to the job's record.

## See Also

**Change Queue Job Position 0x2222 23 110 (page 763), Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Job File Size (old) 0x2222 23 120 (page 786)**

# Change Queue Job Position 0x2222 23 110

Changes a job's position in a queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (8)	word (Hi-Lo)
9	SubFunctionCode (110)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)
16	NewPosition	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
208	0xD0	Queue Error
209	0xD1	No Queue
213	0xD5	No Queue Job
214	0xD6	No Job Right
254	0xFE	Bindery Locked
255	0xFF	Bindery Failure

## Remarks

Position 1 is the first position in a queue and position 250 is the last position in a full queue. If an operator specifies a position number higher than the position number of the job that is currently at the end of the queue, the job is placed at the end of the queue. When a job is moved, the job positions of all job entries in the queue are updated to reflect the new position.

## See Also

**Change Queue Job Entry (old) 0x2222 23 109 (page 761), Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Job File Size (old) 0x2222 23 120 (page 786)**



# Change To Client Rights 0x2222 23 133

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (133)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Restore Queue Server Rights 0x2222 23 117 (page 808)**

# Change To Client Rights (old) 0x2222 23 116

Changes a server's login identity to match that of the client.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (116)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure, File Exists Error

## Remarks

Only a queue server that has previously accepted a job for service can call **Change To Client Rights (old)**.

**Change To Client Rights (old)** replaces the server's login user ID and associated security equivalence list with the ID and security equivalence list of the object (user) that placed the job in the job queue.

**Change To Client Rights (old)** does not change any queue server path mappings (directory bases) on the file server. However, all access rights to those directories are recalculated to conform with the client's rights. Any files opened before **Change To Client Rights (old)** is called continue to be accessible with the server's rights; any files opened after **Change To Client Rights (old)** is called are accessible only with the client's rights. After **Change To Client Rights (old)** is called, the server creates any path mappings it requires to carry out the client's requests.

**Restore Server's Rights 0x2222 23 117** can reverse the effects of **Change To Client Rights (old)**. Also, the server's rights are automatically reset if the server calls **Finish Servicing Queue Job** or **Abort Servicing Queue Job**.

## See Also

**Restore Queue Server Rights 0x2222 23 117 (page 808)**

# Close File And Start Queue Job 0x2222 23 127

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (127)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Create Queue Job And File 0x2222 23 121 (page 773), Abort Servicing Queue Job 0x2222 23 132 (page 754), Finish Servicing Queue Job 0x2222 23 131 (page 782)**

# Close File And Start Queue Job (old) 0x2222 23 105

Closes a client's queue job and marks it ready for execution.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (105)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Lo-Hi)

## Parameters

*JobNumber*

(Request) Specifies the number that the queue management process assigned to the job when the job first entered the queue.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active

Decimal	Hex	Description
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure, File Exists Error

## Remarks

Only the client that created the queue job can call **Close File And Start Queue Job (old)**.

**Close File And Start Queue Job (old)** performs the following actions:

- ♦ Closes the queue file associated with the job
- ♦ Resets the entry open flag (in the job control flags)
- ♦ Marks the entry ready for service (if the user hold and operator hold flags are both clear)

## See Also

**Create Queue Job And File (old) 0x2222 23 104 (page 775), Abort Servicing Queue Job (old) 0x2222 23 115 (page 755), Finish Servicing Queue Job (old) 0x2222 23 114 (page 783)**

# Create Queue 0x2222 23 100

Creates a new queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (100)	byte
10	QueueType	word (Hi-Lo)
12	QueueNameLen	byte
13	QueueName	byte[QueueNameLen]
13 + QueueNameLen	PathBase	byte
14 + QueueNameLen	PathLen	byte
15 + QueueNameLen	PathName	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	QueueID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server

Decimal	Hex	Description
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
238	0xEE	Object Already Exists
255	0xFF	Failure

## Remarks

Only a station with supervisor privileges can call **Create Queue**.

**Create Queue** creates the following:

- ♦ A queue (of type *QueueType*) with the name specified by *QueueName* in the file server's bindery
- ♦ The Q\_DIRECTORY property (with a path set to the directory specified by *PathBase* as modified by *PathName*)
- ♦ A new subdirectory whose name is the 8-character ASCII hexadecimal representation of the new queue's bindery object ID number (Q\_DIRECTORY is initialized with the path name of this newly created directory)
- ♦ The Q\_SERVERS, Q\_OPERATORS, and Q\_USERS group properties (with no members)

*QueueName* can be no longer than 47 characters. The directory's path name (as specified by *PathBase* and *PathName*) can be no longer than 118 characters. A *PathBase* of zero, a *PathLength* of 10, and a *PathName* string of "SYS:SYSTEM" are conventional.

**Create Queue** is actually accomplished by Bindery Services functions. The main advantage of using **Create Queue** is that the entire operation is backed out if any step of the creation process fails.

$SubFuncStrucLen = 6 + QueueNameLen + PathLen.$

## See Also

**Destroy Queue 0x2222 23 101 (page 778)**



# Create Queue Job And File 0x2222 23 121

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + JobStruct)	word (Hi-Lo)
9	SubFunction (121)	byte
10	QueueID	long (Hi-Lo)
12	JobStruct (page 826)	structure

## Reply Format

Offset	Content	Type
Reply header		
8	RecordInUseFlag	word (Lo-Hi)
10	PreviousRecord	long (Lo-Hi)
14	NextRecord	long (Lo-Hi)
18	ClientStation	long (Lo-Hi)
22	ClientTask	long (Lo-Hi)
26	ClientIDNumber	long (Hi-Lo)
30	TargetServerIDNum	long (Hi-Lo)
34	TargetExecutTime[6]	byte
40	JobEntryTime[6]	byte
46	JobNumber	long (Lo-Hi)
50	JobType	word (Lo-Hi)
52	JobPosition	word (Lo-Hi)
54	JobControlFlags	word (Lo-Hi)
56	JobFileName[14]	byte

Offset	Content	Type
70	JobFileHandle	long (Hi-Lo)
74	ServerStation	long (Lo-Hi)
78	ServerTask	long (Lo-Hi)
82	ServerIDNumber	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Close File And Start Queue Job 0x2222 23 127 (page 768), Finish Servicing Queue Job 0x2222 23 131 (page 782)**

# Create Queue Job And File (old) 0x2222 23 104

Creates a new job and enters it into a queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (261)	word (Hi-Lo)
9	SubFunctionCode (104)	byte
10	QueueID	long (Hi-Lo)
14	Job	<a href="#">JobStruct (page 826)</a>

## Reply Format

Offset	Content	Type
Reply header		
8	ClientStation	byte
9	ClientTask	byte
10	ClientIDNumber	long (Hi-Lo)
14	TargetServerIDNumber	long (Hi-Lo)
18	TargetExecutionTime	byte[6]
24	JobEntryTime	byte[6]
30	JobNumber	word (Hi-Lo)
32	JobType	word (Hi-Lo)
34	JobPosition	byte
35	JobControlFlags	byte
36	JobFileName	byte[14]
50	JobFileHandle	byte[6]
56	ServerStation	byte
57	ServerTask	byte

Offset	Content	Type
58	ServerIDNumber	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
252	0xFC	No Such Object
255	0xFF	Failure

## Remarks

Any station that is security equivalent to one of the objects listed in the target queue's Q\_USER property can call **Create Queue Job And File (old)**.

You must provide both the queue ID number to which the job should be appended and the entire 256-byte job record.

The job record fields that you must supply are:

TargetServerIDNumber

TargetExecutionTime

JobType

JobControlFlags (Operator Hold, User Hold, Service Restart, Auto Start)

TextJobDescription

ClientRecordArea

The queue management process fills in the job record and returns it (minus *TextJobDescription* and *ClientRecordArea*) to the calling station. Specifically, queue management creates a file whose name and handle are contained in *JobFileName* and *JobFileHandle*. The calling station can place information (commands, text, etc.) destined for the queue server in this file.

A client using a DOS workstation can open the NETQ device to attach the returned handle to the DOS file created by **Create Queue Job And File (old)**.

## See Also

**Close File And Start Queue Job (old) 0x2222 23 105 (page 769)**

# Destroy Queue 0x2222 23 101

Destroys the specified queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (101)	byte
10	QueueID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Only a station with supervisor privileges can call **Destroy Queue**.

**Destroy Queue** performs the following actions:

- ♦ Aborts all active jobs
- ♦ Detached all server from the queue
- ♦ Destroys all jobs in the queue
- ♦ Deletes all files associated with the queue
- ♦ Removes the queue object and its associated properties from the bindery
- ♦ Deletes the queue's subdirectory

## See Also

**Create Queue 0x2222 23 100 (page 771)**

# Detach Queue Server From Queue 0x2222 23 112

Removes the calling station from a specified queue's list of active queue servers.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (112)	byte
10	QueueID	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure



## Remarks

Only a station that has previously attached itself to the queue as a server can call **Detach Queue Server From Queue**.

A job's service is automatically aborted if the calling station is servicing a job.

## See Also

[\*\*Attach Queue Server To Queue 0x2222 23 111 \(page 757\)\*\*](#)

# Finish Servicing Queue Job 0x2222 23 131

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFunction (131)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)
18	ChargeInfo	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Close File And Start Queue Job 0x2222 23 127 (page 768), Create Queue Job And File 0x2222 23 121 (page 773), Abort Servicing Queue Job 0x2222 23 132 (page 754)**

# Finish Servicing Queue Job (old) 0x2222 23 114

Destroys the job entry, closes and deletes the job file, and restores the calling server's access rights to the file server to their original login values.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunctionCode (114)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)
16	ChargeInformation	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers

Decimal	Hex	Description
255	0xFF	Failure

## Remarks

Only a queue server that has previously accepted a job for service can call **Finish Servicing Queue Job (old)**.

**Finish Servicing Queue Job (old)** signals the queue management process that a queue server has successfully serviced a job.

## See Also

**Close File And Start Queue Job (old) 0x2222 23 105 (page 769), Create Queue Job And File (old) 0x2222 23 104 (page 775), Abort Servicing Queue Job (old) 0x2222 23 115 (page 755)**

# Get Queue Job File Size 0x2222 23 135

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (135)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	QueueID	long (Hi-Lo)
12	JobNumber	long (Lo-Hi)
16	FileSize	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Get Queue Job List 0x2222 23 129 (page 788), Read Queue Current Status 0x2222 23 125 (page 794), Read Queue Job Entry 0x2222 23 122 (page 799), Remove Job From Queue 0x2222 23 128 (page 805)**

# Get Queue Job File Size (old) 0x2222 23 120

Returns the current length of a file associated with a queue entry.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (120)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	QueueID	long (Hi-Lo)
12	JobNumber	long (Lo-Hi)
16	FileSize	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job

Decimal	Hex	Description
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Any station that is security equivalent to an object listed in the queue's Q\_USERS group property or Q\_OPERATORS group property can call **Get Queue Job File Size (old)**.

If the queue entry is still open (not committed by its creator) or is in service, the returned entry size does not necessarily reflect the final file size.

## See Also

**Get Queue Job List (old) 0x2222 23 107 (page 790), Read Queue Current Status (old) 0x2222 23 102 (page 796), Read Queue Job Entry (old) 0x2222 23 108 (page 800), Remove Job From Queue (old) 0x2222 23 106 (page 806)**

# Get Queue Job List 0x2222 23 129

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (129)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	TotalQueueJobs	long (Lo-Hi)
12	ReplyQueueJobNumbers	long (Lo-Hi)
16	JobNumberList[]	long (Lo-Hi)

## Parameters

*TotalQueueJobs*

(Reply) Specifies the number of jobs in the queue.

*ReplyQueueJobNumbers*

(Reply) Specifies the number of jobs in *JobNumberList* (maximum is 125).

## Return Values

Decimal	Hex	Description
0	0x00	Successful



## See Also

**Get Queue Job File Size 0x2222 23 135 (page 785), Read Queue Current Status 0x2222 23 125 (page 794), Read Queue Job Entry 0x2222 23 122 (page 799), Remove Job From Queue 0x2222 23 128 (page 805)**

# Get Queue Job List (old) 0x2222 23 107

Returns the list of jobs that are contained in the specified queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (107)	byte
10	QueueID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	JobCount	word (Hi-Lo)
10	JobNumber	word[JobCount] (Hi-Lo)

## Parameters

*JobCount*

(Reply) Specifies the number of entries that are currently in the queue.

*JobNumber*

(Reply) Specifies the list of entries in their current queue order.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue

Decimal	Hex	Description
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Any station that is security equivalent to an object listed in the queue's Q\_USERS group property or Q\_OPERATORS group property can call **Get Queue Job List (old)**.

When used in conjunction with **Read Queue Job Entry** 0x2222 108, **Get Queue Job List (old)** returns the following information about all the jobs in the specified queue:

- ◆ Each job's position in the queue.
- ◆ The number and type of jobs in the queue. (This changes between a station's consecutive requests because the queue management environment is multi-threaded.)
- ◆ The job number of each job in the queue.

If your request to read information about a job in the queue fails with No Queue Job, you can assume that the job has been completed or that it has been deleted from the queue.

## See Also

**Get Queue Job File Size (old) 0x2222 23 120 (page 786), Read Queue Current Status (old) 0x2222 23 102 (page 796), Read Queue Job Entry (old) 0x2222 23 108 (page 800), Remove Job From Queue (old) 0x2222 23 106 (page 806)**

# Get Queue Jobs From Form List 0x2222 23 137

Returns a list of jobs that are in a queue, using a form-type match list.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (15)	word (Hi-Lo)
9	SubFuncCode (137)	byte
10	QueueID	long (Hi-Lo)
14	QueueStartPosition	long (Lo-Hi)
18	FormTypeCnt	long (Lo-Hi)
22	FormTypeList	word[FormTypeCnt] (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	TotalQueueJobs	long (Lo-Hi)
12	pkQJobCnt	long (Lo-Hi)
16	JobNumberList	long[pkQJobCnt] (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Bad Length
252	0xFC	No Such Object

## See Also

[Get Queue Job List 0x2222 23 129 \(page 788\)](#), [Get Queue Job List \(old\) 0x2222 23 107 \(page 790\)](#)

# Move Queue Job From Src Q to Dst Q 0x2222 23 136

Moves a queue job from a source queue to a destination queue and returns a new job number.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (136)	byte
10	SrcQueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)
18	DstQueueID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	NewJobNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Bad Length
252	0xFC	No Such Object

## See Also

**Get Queue Job List 0x2222 23 129 (page 788), Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Jobs From Form List 0x2222 23 137 (page 792)**

# Read Queue Current Status 0x2222 23 125

Allows the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunction (125)	byte
10	QueueID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	QueueID	long (Hi-Lo)
12	QueueStatus	long (Lo-Hi)
16	CurrentEntries	long (Lo-Hi)
20	CurrentServers	long (Lo-Hi)
24	ServerIDList[]	long (Lo-Hi)
xx	ServerStationList[]	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights

Decimal	Hex	Description
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

*ServerIDList* and *ServerStationList* are arrays. The number of entries in both arrays are determined by *CurrentServers*.

## See Also

**Get Queue Job List 0x2222 23 129 (page 788), Get Queue Job File Size 0x2222 23 135 (page 785), Read Queue Job Entry 0x2222 23 122 (page 799), Remove Job From Queue 0x2222 23 128 (page 805), Set Queue Current Status 0x2222 23 126 (page 819)**

# Read Queue Current Status (old) 0x2222 23 102

Returns the current status of a queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (102)	byte
10	QueueID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	QueueID	long (Hi-Lo)
12	QueueStatus	byte
13	CurrentEntries	byte
14	CurrentServers	byte
15	ServerIDList[]	long[CurrentServers] (Hi-Lo)
15 + CurrentServers*4	ServerStationList[]	byte[CurrentServers]

## Parameters

### *QueueStatus*

(Reply) Specifies the overall status of the queue:

- 1 The operator does not want to add jobs to the queue
- 2 The operator does not want additional servers attaching to the queue
- 4 The operator does not want servers to service jobs in the queue

### *CurrentEntries*

(Reply) Specifies the number of jobs that are currently in the queue (0 to 250).



### *CurrentServers*

(Reply) Specifies the number of currently attached servers that can service this queue (0 to 25).

### *ServerIDList*

(Reply) Specifies the servers that are currently servicing the queue by object ID number.

### *ServerStationList*

(Reply) Specifies the servers that are currently servicing the queue by current station attachment.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Any station that is security equivalent to any object listed in the queue's Q\_USERS group property or Q\_OPERATORS group property can call **Read Queue Current Status (old)**.

## See Also

**Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Job File Size 0x2222 23 135 (page 785), Read Queue Job Entry (old) 0x2222 23 108 (page 800), Remove Job From Queue (old) 0x2222 23 106 (page 806)**

# Read Queue Job Entry 0x2222 23 122

Enables the use of the high connection byte in the request and reply header of the packet.

NetWare Server: 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (122)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	JobStruct (page 826)	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

Get Queue Job List 0x2222 23 129 (page 788), Get Queue Job File Size 0x2222 23 135 (page 785), Read Queue Current Status 0x2222 23 125 (page 794), Remove Job From Queue 0x2222 23 128 (page 805)

# Read Queue Job Entry (old) 0x2222 23 108

Returns the full 256-byte queue job record for the specified queue ID and job number.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (108)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	Job	JobStruct (page 826)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing

Decimal	Hex	Description
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Any station that is security equivalent to an object listed in the queue's Q\_USERS group property or Q\_OPERATORS group property can call **Read Queue Job Entry (old)**.

## See Also

**Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Job File Size (old) 0x2222 23 120 (page 786), Read Queue Current Status (old) 0x2222 23 102 (page 796), Remove Job From Queue (old) 0x2222 23 106 (page 806)**

# Read Queue Server Current Status 0x2222 23 134

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFunction (134)	byte
10	QueueID	long (Hi-Lo)
14	ServerID	long (Hi-Lo)
18	ServerStation	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ServerStatusRecord[64]	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Read Queue Server Current Status (old) 0x2222 23 118

Returns the current status of a queue server.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (10)	word (Hi-Lo)
9	SubFunctionCode (118)	byte
10	QueueID	long (Hi-Lo)
14	ServerID	long (Hi-Lo)
18	ServerStation	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ServerStatusRecord	byte[64]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right

Decimal	Hex	Description
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Any station that is security equivalent to one of the objects listed in the queue's Q\_USERS group property or Q\_OPERATORS group property can call **Read Queue Server Current Status (old)**.

The queue management process in the file server maintains a 64-byte status record for each queue server that is attached to a queue. The information in the *ServerStatusRecord* can be anything of use to queue servers and queue clients; this record is not used or interpreted by the queue manager.

The first 4 bytes of *ServerStatusRecord* is the estimated price for the given queue server to complete an average job.

## See Also

**Set Queue Server Current Status 0x2222 23 119 (page 822)**



# Remove Job From Queue 0x2222 23 128

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (128)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Get Queue Job List 0x2222 23 129 (page 788), Read Queue Current Status 0x2222 23 125 (page 794), Read Queue Job Entry 0x2222 23 122 (page 799), Get Queue Job File Size 0x2222 23 135 (page 785)**

# Remove Job From Queue (old) 0x2222 23 106

Removes a job from a job queue.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (106)	byte
10	QueueID	long (Hi-Lo)
14	JobNumber	word (Hi-Lo)

## Parameters

*JobNumber*

(Reply) Specifies the number that the queue manager assigned to the job when it was first entered into the queue.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active

Decimal	Hex	Description
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Only an operator or the client that created the job can call **Remove Job From Queue (old)**.

**Remove Job From Queue (old)** performs the following actions:

- ♦ Removes the specified job from the queue
- ♦ Closes and deletes the associated file
- ♦ Aborts the service if the job is in the process of being serviced

Any further I/O requests made by the server to the job's queue file fail, and Illegal File Handle is returned.

## See Also

**Get Queue Job List (old) 0x2222 23 107 (page 790), Read Queue Current Status (old) 0x2222 23 102 (page 796), Get Queue Job File Size (old) 0x2222 23 120 (page 786)**

# Restore Queue Server Rights 0x2222 23 117

Restores a queue server's identity after it has assumed its client's identity by calling **Change To Client Rights** 0x2222 23 116.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (117)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Only a queue server that has previously changed its identity by calling **Change To Client Rights** can call **Restore Queue Server Rights**.

The queue server's login user ID and the associated security equivalence list are also restored when **Restore Queue Server Rights** completes.

**Restore Queue Server Rights** does not change any queue server path mappings (directory bases) on the file server. However, all access rights to those directories are recalculated to conform with the queue server's rights. If the queue server changes some of its path mappings in order to service the queue job, the queue restores its own directory bases.

Any files that are opened before **Restore Queue Server Rights** is called continue to be accessible with the rights of the queue client; any files that are opened after **Restore Queue Server Rights** is called are accessible to any station with the rights of the queue server.

## See Also

**Change To Client Rights 0x2222 23 133 (page 765)**

# Service Queue Job 0x2222 23 124

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunction (124)	byte
10	QueueID	long (Hi-Lo)
14	TargetServiceType	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	RecordInUseFlag	word (Lo-Hi)
10	PreviousRecord	long (Lo-Hi)
14	NextRecord	long (Lo-Hi)
18	ClientStation	long (Lo-Hi)
22	ClientTask	long (Lo-Hi)
26	ClientIDNumber	long (Hi-Lo)
30	TargetServerIDNum	long (Hi-Lo)
34	TargetExecutTime[6]	byte
40	JobEntryTime[6]	byte
46	JobNumber	long (Lo-Hi)
50	JobType	word (Lo-Hi)
52	JobPosition	word (Lo-Hi)
54	JobControlFlags	word (Lo-Hi)
56	JobFileName[14]	byte

Offset	Content	Type
70	JobFileHandle	long (Lo-Hi)
74	ServerStation	long (Lo-Hi)
78	ServerTask	long (Lo-Hi)
82	ServerIDNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Service Queue Job (old) 0x2222 23 113

Selects a new job for servicing.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (113)	byte
10	QueueID	long (Hi-Lo)
14	TargetServiceType	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	ClientStation	byte
9	ClientTask	byte
10	ClientIDNumber	long (Hi-Lo)
14	TargetServerIDNumber	long (Hi-Lo)
18	TargetExecutionTime	byte[6]
24	JobEntryTime	byte[6]
30	JobNumber	word (Hi-Lo)
32	JobType	word (Hi-Lo)
34	JobPosition	byte
35	JobControlFlags	byte
36	JobFileName	byte[14]
50	JobFileHandle	byte[6]
56	ServerStation	byte
57	ServerTask	byte



Offset	Content	Type
58	ServerIDNumber	long (Hi-Lo)

## Parameters

*TargetServerIDNumber*

(Reply) Specifies the ID number of the calling queue server (or -1L).

*TargetExecutionTime*

(Reply) Specifies an earlier time than the time indicated on the current system clock (or 0xFF).

*JobType*

(Reply) Specifies the server's specified *TargetServiceType* (or *TargetServiceType* must be -1).

*ServerIDNumber*

(Reply) Specifies that the job is not currently being serviced by some other server (zero).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Only a station that has previously attached itself to the specified queue as a server can call **Service Queue Job**.

All jobs in the queue are searched to find the one that meets the criteria outlined in the Parameters section. Also, Operator Hold, User Hold, and Entry Open Flags in *JobControlFlags* must all be reset to zero.

If a job meets all of the above criteria, the job is marked for servicing by the calling station as follows:

- ♦ The servicing station ID is entered into the job's *ServerStation*, *ServerTask*, and *ServerIDNumber* values.
- ♦ The job file is opened for read and write access by the server.
- ♦ The updated job entry record is delivered to the calling server for service.

## See Also

**Attach Queue Server To Queue 0x2222 23 111 (page 757)**

# Service Queue Job By Form List 0x2222 23 138

Retrieves a job for servicing.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (11)	word (Hi-Lo)
9	SubFuncCode (138)	byte
10	QueueID	long (Hi-Lo)
14	FormTypeCnt	long (Lo-Hi)
18	FormTypeList	word[FormTypeCnt] (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	QRecordInUse	word (Lo-Hi)
10	QPreviousRecord	long (Lo-Hi)
14	QNextRecord	long (Lo-Hi)
18	QStation	long (Lo-Hi)
22	QTask	long (Lo-Hi)
26	QUserID	long (Hi-Lo)
30	TargetServerID	long (Hi-Lo)
34	TargetTime[6]	byte[6]
40	QEntryTime[6]	byte[6]
46	QJobNumber	long (Lo-Hi)
50	QEntryType	word (Lo-Hi)
52	QOrder	word (Lo-Hi)
54	QFlags	word (Lo-Hi)

Offset	Content	Type
56	QFileName	byte[14]
70	QFileHandle	long (Lo-Hi)
74	QServerStation	long (Lo-Hi)
78	QServerTask	long (Lo-Hi)
82	QServerID	long (Hi-Lo)

## Parameters

### *QRecordInUse*

Specifies whether the record is in use.

### *QPreviousRecord*

Points to the previous record.

### *QNextRecord*

Points to the next record.

### *QStation*

Specifies the connection number of the station that placed the job in the queue.

### *QTask*

Specifies the number of the task that was active when the job was placed in the queue, as filled by QMS.

### *QUserID*

Specifies the object ID number of the user that is logged in to the station making the queue entry, as filled in by QMS.

### *TargetServerID*

Specifies the ID number of the server that the client specifies to service this entry.

### *TargetTime*

Specifies the earliest time that the queue client is willing to have the job serviced (year, month, day, hour, minute, second or 0xFFFFFFFF for the first opportunity).

### *QEntryTime*

Specifies the time that the job entered the queue, as filled in by QMS (year, month, day, hour, minute, and second from the server's system clock).

### *QJobNumber*

Specifies the unique number that servers or clients can use when referring this queue entry, as assigned by WMS when the job is created.

### *QEntryType*

Specifies the type of job represented by the queue entry.

### *QOrder*

Specifies the job's position within the queue.

### *QFlags*

Specifies the current status of the job:

---

0x08	The queue job should automatically be started even if the client's connection is broken (the station is turned off or the client is logged out) and before the client signals that the job is ready to be queued. If this bit is cleared, the job will be removed from the queue if the creating station destroys its connection with the file server before signalling that the entry is ready for service. If this bit is set, the job will automatically be marked ready for service when a station's connection is lost. This bit can be set or cleared by the client that created the entry or by an operator.
0x10	The queue entry is to be reserviced if the server fails. A server can indicate that it was unable to finish servicing a job and has aborted service. Also, servers that break their connection with the file server without signaling successful completion of service on outstanding jobs are assumed to have aborted service on those jobs. If service is aborted for an entry that has this bit set, the entry will remain in its same position in the service queue and will be readied for reservice at a later time. If service is aborted for an entry that has this bit cleared, the entry will be removed from the queue and will not be reserviced. This bit can be set or cleared by the client that created the entry or by an operator.
0x20	When an entry is first created, this bit is set to indicate that the file associated with the entry has not yet been closed by the client. When the client indicates that it has finished preparing the queue entry and the queue file, the file is closed and this bit is cleared. This bit is managed by QMS.
0x40	This bit is set if the entry has been placed on hold by the client that created the entry. Jobs placed on hold continue to advance toward the front of the queue, but they will not be serviced until the hold bit has been cleared. This bit can be set or cleared by the client that created the entry or by an operator.
0x80	This bit is set if the entry has been placed on hold by an operator. Jobs placed on hold continue to advance towards the front of the queue, but they will not be serviced until the hold bit has been cleared. This bit can be set or cleared only by an operator.

---

### *QFileName*

Specifies the name of the file that holds data associated with the queue entry.

### *QFileHandle*

Specifies the file handle (in standard NetWare format) that a job client or job server can use to access the file associated with an opened job.

### *QServerStation*

Specifies the station number of the queue server that is servicing the job, as maintained by QMS.

### *QServerTask*

Specifies the task number of the queue server that is servicing the job, as maintained by QMS.

### *QServerID*

Specifies the ID number of the queue server that is servicing the job, as maintained by QMS.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Bad Length
255	0xFF	Failure

## Remarks

The server specifies the job queue and the entry type list that it wants to service. If the server is attached to the target queue and the queue has one or more jobs that match the requirements, the first such job in the queue is given to the calling server and the file associated with the queue entry is automatically opened.

Set *TargetServerID* to -1L if this entry can be serviced by any server. If the specified server is not authorized to service the queue, the job is rejected. *TargetServerID* must be supplied by the client that created the queue job.

A server can specify that it will service only the entries for particular types. Then a queue can be used to hold entries for job requiring different types of service. *QEntryType* must be supplied by the client that created the queue job. Do not use -1 as the entry type.

The first job will be placed at the head of the queue and will be assigned position number 1; the second job will be placed behind the first job and will be assigned position number 2, etc. As jobs are removed from the queue, the position numbers of the remaining jobs are updated to reflect their changing position within the queue. The job position number is assigned and maintained by QMS.

When a client creates a queue entry, QMS creates a file with a unique name in the queue directory and grants the queue read and write access to the file.

NetWare shells provide hooks to allow applications to open the special NETQ device after opening a queue job to attach the file handle specified by *QFileHandle* to a DOS file. Applications can then use standard DOS requests to access the job file. *QFileHandle* is set by QMS.

## See Also

**[Get Queue Job List 0x2222 23 129 \(page 788\)](#), [Get Queue Job List \(old\) 0x2222 23 107 \(page 790\)](#), [Get Queue Jobs From Form List 0x2222 23 137 \(page 792\)](#)**

# Set Queue Current Status 0x2222 23 126

Enables the use of the high connection byte in the request and reply header of the packet.

**NetWare Server:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunction (126)	byte
10	QueueID	long (Hi-Lo)
14	QueueStatus	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

**Get Queue Job List 0x2222 23 129 (page 788), Get Queue Job File Size 0x2222 23 135 (page 785), Read Queue Job Entry 0x2222 23 122 (page 799), Remove Job From Queue 0x2222 23 128 (page 805), Read Queue Current Status 0x2222 23 125 (page 794)**

# Set Queue Current Status (old) 0x2222 23 103

Allows an operator to control a queue's status.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (103)	byte
10	QueueID	long (Hi-Lo)
14	QueueStatus	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
252	0xFC	No Such Object



Decimal	Hex	Description
255	0xFF	Failure

## Remarks

Only a station with operator privileges can call **Set Queue Current Status (old)**.

An operator can control the adding of jobs and adding servers to a queue by setting bits in *QueueStatus* as follows:

- 1 Prevent jobs from being added to the queue
- 2 Prevent servers from attaching to the queue in order to service it
- 4 Prevent servers from servicing jobs in the queue

## See Also

**Get Queue Job List (old) 0x2222 23 107 (page 790), Get Queue Job File Size (old) 0x2222 23 120 (page 786), Remove Job From Queue (old) 0x2222 23 106 (page 806), Read Queue Current Status (old) 0x2222 23 102 (page 796)**

# Set Queue Server Current Status 0x2222 23 119

Allows a queue server to update the queue manager's copy of the queue server's *ServerStatusRecord*.

**NetWare Server:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (69)	word (Hi-Lo)
9	SubFunctionCode (119)	byte
10	QueueID	long (Hi-Lo)
14	ServerStatusRecord	byte[64]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
153	0x99	Directory Full Error
208	0xD0	Queue Error
209	0xD1	No Queue
210	0xD2	No Queue Server
211	0xD3	No Queue Rights
212	0xD4	Queue Full
213	0xD5	No Queue Job
214	0xD6	No Job Right
215	0xD7	Queue Servicing
216	0xD8	Queue Not Active
217	0xD9	Station Not Server
218	0xDA	Queue Halted
219	0xDB	Maximum Queue Servers
255	0xFF	Failure

## Remarks

Only a station that has previously attached to the specified queue as a queue server can call **Set Queue Server Current Status**.

The first 4 bytes of ServerStatusRecord should contain the estimated cost for the queue server to complete an average job of the type it services. The queue manager does not interpret the contents of this status record so its format should be agreed upon by the queue server and queue clients.

## See Also

**[Read Queue Server Current Status 0x2222 23 134 \(page 802\)](#)**



# 36 Structures

This section describes the Queue structures and their fields.

# JobStruct

Contains job control information for queues.

## Syntax (prior to NetWare 3.x)

Offset	Content	Type
0	ClientStation	byte
1	ClientTaskNumber	byte
2	ClientIDNumber	long (Hi-Lo)
6	TargetServerIDNumber	long (Hi-Lo)
10	TargetExecutionTime	byte[6]
16	JobEntryTime	byte[6]
22	JobNumber	word (Hi-Lo)
24	JobType	word (Hi-Lo)
26	JobPosition	byte
27	JobFileName	byte[14]
42	JobFileHandle	byte
48	ServerStation	byte
49	ServerTaskNumber	byte
50	ServerIDNumber	long (Hi-Lo)
54	TextJobDescription	byte[50]
104	ClientRecordArea	byte[152]

## Syntax (NetWare 3.x and later)

Offset	Content	Type
0	RecordInUse	word (Lo-Hi)
2	PreviousRecord	long (Lo-Hi)
6	NextRecord	long (Lo-Hi)
10	ClientStation	long (Lo-Hi)
14	ClientTaskNumber	long (Lo-Hi)
18	ClientIDNumber	long (Hi-Lo)

Offset	Content	Type
22	TargetServerIDNumber	long (Hi-Lo)
26	TargetExecutionTime	byte[6]
32	JobEntryTime	byte[6]
38	JobNumber	word (Lo-Hi)
42	JobType	word (Lo-Hi)
44	JobPosition	word (Lo-Hi)
46	JobControlFlags	word (Lo-Hi)
48	JobFileName	byte[14]
62	JobFileHandle	long (Hi-Lo)
66	ServerStation	long (Lo-Hi)
70	ServerTaskNumber	long (Lo-Hi)
74	ServerIDNumber	long (Hi-Lo)
78	TextJobDescription	byte[50]
128	ClientRecordArea	byte[152]

## Parameters

### *RecordInUse*

(NetWare 3.0 and later) Specifies whether the record is in use.

### *PreviousRecord*

(NetWare 3.0 and later) Points to the previous record.

### *NextRecord*

(NetWare 3.0 and later) Points to the next record.

### *ClientStation*

Specifies the connection number of the station that placed the job in the queue.

### *ClientTaskNumber*

Specifies the number of the task that was active when the job was placed in the queue, as filled by QMS.

### *ClientIDNumber*

Specifies the object ID number of the user that is logged in to the station making the queue entry, as filled in by QMS.

### *TargetServerID*

Specifies the ID number of the server that the client specifies to service this entry.

### *TargetExecutionTime*

Specifies the earliest time that the queue client is willing to have the job serviced (year, month, day, hour, minute, second or 0xFFFFFFFF for the first opportunity).

### *JobEntryTime*

Specifies the time that the job entered the queue, as filled in by QMS (year, month, day, hour, minute, and second from the server's system clock).

### *JobNumber*

Specifies the unique number that servers or clients can use when referring this queue entry, as assigned by QMS when the job is created.

### *JobType*

Specifies the type of job represented by the queue entry.

### *JobPosition*

Specifies the job's position within the queue.

### *JobControlFlags*

Specifies the current status of the job:

---

0x08	The queue job should automatically be started even if the client's connection is broken (the station is turned off or the client is logged out) and before the client signals that the job is ready to be queued. If this bit is cleared, the job will be removed from the queue if the creating station destroys its connection with the file server before signalling that the entry is ready for service. If this bit is set, the job will automatically be marked ready for service when a station's connection is lost. This bit can be set or cleared by the client that created the entry or by an operator.
0x10	The queue entry is to be reserviced if the server fails. A server can indicate that it was unable to finish servicing a job and has aborted service. Also, servers that break their connection with the file server without signaling successful completion of service on outstanding jobs are assumed to have aborted service on those jobs. If service is aborted for an entry that has this bit set, the entry will remain in its same position in the service queue and will be readied for reservice at a later time. If service is aborted for an entry that has this bit cleared, the entry will be removed from the queue and will not be reserviced. This bit can be set or cleared by the client that created the entry or by an operator.
0x20	When an entry is first created, this bit is set to indicate that the file associated with the entry has not yet been closed by the client. When the client indicates that it has finished preparing the queue entry and the queue file, the file is closed and this bit is cleared. This bit is managed by QMS.
0x40	This bit is set if the entry has been placed on hold by the client that created the entry. Jobs placed on hold continue to advance toward the front of the queue, but they will not be serviced until the hold bit has been cleared. This bit can be set or cleared by the client that created the entry or by an operator.
0x80	This bit is set if the entry has been placed on hold by an operator. Jobs placed on hold continue to advance towards the front of the queue, but they will not be serviced until the hold bit has been cleared. This bit can be set or cleared only by an operator.

---



### *JobFileName*

Specifies the name of the file (in DOS 8.3 format) that holds data associated with the queue entry.

### *JobFileHandle*

Specifies the file handle (in standard NetWare format) that a job client or job server can use to access the file associated with an opened job.

### *ServerStation*

Specifies the station number of the queue server that is servicing the job, as maintained by QMS.

### *ServerTaskNumber*

Specifies the task number of the queue server that is servicing the job, as maintained by QMS.

### *ServerIDNumber*

Specifies the ID number of the queue server that is servicing the job, as maintained by QMS.

### *TextJobDescription*

Specifies a NULL-terminated ASCII text description of the contents or purpose of the job, which can be displayed by QMS as part of the job description.

### *ClientRecordArea*

Specifies supplementary information that can be exchanged between the queue client and the queue server.

## Remarks

A queue can hold from 0 to 250 jobs. Jobs are numbered from 1 to 999.

A queue can be serviced by as many as 25 job servers.

The job structure for NetWare versions prior to 3.x has a fixed length of 256 bytes.

When a new job is placed in a queue, the file server automatically creates a file for this station and gives the station read and write access to the file. The queue management process in that file server sets the *ClientStation* field.

Set *TargetServerID* to -1L if this entry can be serviced by any server. If the specified server is not authorized to service the queue, the job is rejected. *TargetServerID* must be supplied by the client that created the queue job.

A server can specify that it will service only the entries for particular job types. Then a queue can be used to hold entries for job requiring different types of service. *JobType* must be supplied by the client that created the queue job. Do not use -1 as the job type.

The first job will be placed at the head of the queue and will be assigned position number 1; the second job will be placed behind the first job and will be assigned position number 2, etc. As jobs are removed from the queue, the position numbers of the remaining jobs are updated to reflect their changing position within the queue. The job position number is assigned and maintained by QMS.

When a client creates a queue entry, QMS creates a file with a unique name in the queue directory and grants the queue read and write access to the file.

NetWare shells provide hooks to allow applications to open the special NETQ device after opening a queue job to attach the file handle specified by *JobFileHandle* to a DOS file. Applications can then use standard DOS requests to access the job file. *JobFileHandle* is set by QMS.

# XV RPC

To access RPC NCPs, use the following:

- ♦ [Chapter 37, “NCPs,” on page 833](#)



# 37

## NCPs

This section describes each of the RPC NCPs, their Request and Reply formats, and Return Values.

# RPC Add Name Space To Volume 0x2222 131 05

Adds a specified name space to a mounted volume on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen (21)	word (Hi-Lo)
9	SubFuncCode (05)	byte
10	reserved (0)	long[4]
26	reservedFlags (0)	byte[4]
30	AddNameSpaceAndVol	ASCIIZ

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long[4]

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *AddNameSpaceAndVol* (in ASCIIZ), including the terminating NULL character.

### *AddNameSpaceAndVol*

(Request) Specifies the name space and volume string (in ASCIIZ format) to add to a selected volume name in the format:

```
"NameSpaceName {TO} {VOLUME} Volume Name\0"
```

**IMPORTANT:** The above format must be followed exactly or the request will fail. The name space should be followed by a space and the volume name (along with the optional "TO VOLUME" keywords). The volume name must not have a colon appended to it.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Invalid Subfunction Request

## RPCccode

0	Successful completion of the request RPC
---	--

## Remarks

The client connection making this request must have logged in to the server, be permanently authenticated, and be at least equivalent to supervisor.

The NameSpaceName values follows:

- 0 DOS
- 1 MAC
- 2 NFS
- 3 FTAM
- 4 OS/2

In NetWare 4.11 and above, OS/2 (OS2.NAM) has been replaced with LONG (LONG.NAM).

# RPC Dismount Volume 0x2222 131 04

Dismounts a volume on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen (21)	word (Hi-Lo)
9	SubFuncCode (04)	byte
10	reserved (0)	long[4]
26	reservedFlags (0)	byte[4]
30	VolumeName	ASCIIZ

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *PathAndName* (in ASCIIZ), including the terminating NULL character.

### *VolumeName*

(Request) Specifies the name of the volume (in ASCIIZ format) to be dismounted in the following format:

```
volume name\0
```



# Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Invalid Subfunction Request

# RPCccode

0	Successful completion of the request RPC
	Invalid Volume Name

# Remarks

The client connection making this request must have logged in to the server, be permanently authenticated, and be at least equivalent to supervisor.

# RPC Execute NCF File 0x2222 131 07

Executes a selected NCF file on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen	word (Hi-Lo)
9	SubFuncCode (07)	byte
10	reserved (0)	long[4]
26	reservedFlags (0)	byte[4]
30	PathAndName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long[4]

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *PathAndName* (in ASCIIZ), including the terminating NULL character.

### *PathAndName*

(Request) Specifies the ASCIIZ path and file name to execute in the format:

```
{volume name:} {path\...}file name\0
```

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
251	0xFB	Invalid Subfunction Request

## RPCccode

0	Successful completion of the request RPC	
158	Bad File Name or No File Name Given	

## Remarks

The client connection making this request must have logged in to the server, be permanently authenticated, and be at least equivalent to supervisor.

# RPC Load an NLM 0x2222 131 01

Loads a selected NLM on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen	word (Hi-Lo)
9	SubFuncCode (01)	byte
10	NLMLoadOptions	long
14	reserved (0)	long[3]
26	reservedFlags (0)	byte[4]
30	PathAndName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long[4]

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *PathAndName* (in ASCIIZ), including the terminating NULL character.

### *NLMLoadOptions*

(Request) Specifies the NLM load options (bit definitions).

### *PathAndName*

(Request) Specifies the ASCIIZ path and file name to load in the following format:

```
{volume name:}{path\...}file name\0
```

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Invalid Subfunction Request

## RPCccode

0	Successful completion of the request RPC
---	--

# RPC Mount Volume 0x2222 131 03

Mounts a volume on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen	word (Hi-Lo)
9	SubFuncCode (03)	byte
10	reserved (0)	long[4]
26	reservedFlags (0)	byte[4]
30	VolumeName	ASCIIZ

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long[4]
28	VolumeNumber (variable)	long

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *PathAndName* (in ASCIIZ), including the terminating NULL character.

### *VolumeName*

(Request) Specifies the name of the volume (in ASCIIZ format) to be mounted in the following format:

```
volume name\0
```

# Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Invalid Subfunction Request

# RPCccode

0	Successful completion of the request RPC
	Invalid Volume Name
	Volume Already Mounted

# RPC Set Set Command Value 0x2222 131 06

Changes the current value of a set command on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen (21)	word (Hi-Lo)
9	SubFuncCode (06)	byte
10	typeFlag	long
14	Value	long
18	reserved (0)	long[2]
26	reservedFlags (0)	byte[4]
30	SetCmdName	ASCIIZ
xx	(optional string)	ASCIIZ

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long[4]

## Parameters

*SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *SetCmdName* (in ASCIIZ), including the terminating NULL character.

*typeFlag*

(Request) Specifies where to find the new value of the set command:

- 0 Set command is in the optional string
- 1 Set command is in *Value*



*Value*

(Request) Specifies the new value of the set command if *typeFlag* is one.

*SetCmdName*

(Request) Specifies the set parameter command name in ASCIIZ format.

*optional string*

(Request) Specifies the new set command value if *typeFlag* is zero.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
251	0xFB	Invalid Subfunction Request

## RPCccode

0	Successful completion of the request RPC
---	--

## Remarks

The client connection making this request must have logged in to the server, be permanently authenticated, and be at least equivalent to supervisor.

If *typeFlag* is zero, the optional string size (including the NULL character) must be included in *SubFuncStrucLen*.

The set command values SP\_TYPE\_STRING and SP\_TYPE\_TIME\_OFFSET require a string instead of a numerical value.

# RPC Unload an NLM 0x2222 131 02

Unloads a selected NLM on the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (131)	byte
7	SubFunctionStrucLen	word (Hi-Lo)
9	SubFuncCode (02)	byte
10	reserved (0)	long[4]
26	reservedFlags (0)	byte[4]
30	NLMName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	RPCccode	long
12	reserved (0)	long

## Parameters

### *SubFuncStrucLen*

(Request) Specifies the length of 21 plus the size of *NLMName* (in ASCIIZ), including the terminating NULL character.

### *NLMName*

(Request) Specifies the ASCIIZ NLM module name to unload in the following format:

```
file name\0
```

## Return Values

Decimal	Hex	Description
0	0x00	Successful

Decimal	Hex	Description
251	0xFB	Invalid Subfunction Request

## RPCccode

0	Successful completion of the request RPC	
158	Bad File Name or No File Name Given	

## Remarks

The client connection making this request must have logged in to the server, be permanently authenticated, and be at least equivalent to supervisor.



# XVI SecretStore

To access SecretStore NCPs, use the following:

- ♦ [Chapter 38, “NCPs,” on page 851](#)



# 38 NCPs

This section describes each of the SecretStore subverbs. For more information, see the section for [Novell SecretStore Developer Kit for C \(http://developer.novell.com/ndk/ssocomp.htm\)](http://developer.novell.com/ndk/ssocomp.htm).

# SecretStore Services 0x2222 92

Contains several subverbs that provide SecretStore functionality.

**NetWare Server:** 5.x

**eDirectory:** 8.5 and later (NT/2000 and Unix/Linux)

## Subverbs

- 
- |   |  |
|---|--|
| 0 | <b>Query Server</b> calls the server, sends the cryptography list that is supported by the client to the server, and requests the cryptography algorithms that are supported by the server. It also send the client's version of Single Sign-on to the server and receives the server version in return. |
| 1 | <b>Read App Secrets</b> sends the Secret ID to the server and, upon success, receives the secret back. Regardless of success or failure, the read returns status information regarding the secret.   |
| 2 | <b>Write App Secrets</b> sends a Secret ID and the secret to the server and receives a confirmation of success or failure back.  |
| 3 | <b>Add Secret ID</b> sends a Secret ID to create an uninitialized secret in the SecretStore.   |
| 4 | <b>Remove Secret ID</b> sends a Secret ID to remove a secret from the SecretStore.   |
| 5 | <b>Remove SecretStore</b> removes the SecretStore from a user's object.  |
| 6 | <b>Enumerate SecretIDs</b> returns a list of secrets from a user's SecretStore.  |
| 7 | <b>Unlock Store</b> unlocks the SecretStore.   |
| 8 | <b>Set Master Password</b> sets a master password on the SecretStore.  |
| 9 | <b>Get Service Information</b> returns status information on the SecretStore.  |
-



# XVII

## Server Environment

Server Environment NCPs enable you to set certain server parameters and return information about servers. You can also retrieve information about the file system, the transaction tracking system, physical disks, disk channels, volumes, disk caches, LAN I/Os, file and record locks, semaphores, connections, and tasks.

Specifically, these NCPs allow you to

- ♦ Enable or disable transaction tracking
- ♦ Prohibit or allow users to log in
- ♦ Set the server time and date
- ♦ Broadcast messages to a group of workstations
- ♦ Clear a connection
- ♦ Down a server

To access Server Environment NCPs, use the following:

- ♦ [Chapter 39, “Concepts,” on page 855](#)
- ♦ [Chapter 40, “NCPs,” on page 857](#)
- ♦ [Chapter 41, “Structures,” on page 967](#)



# 39 Concepts

This section explains ideas that are common to Server Environment NCPs.

## File Access Rights

To safeguard the server from unauthorized access, most Server Environment NCPs require you to have OPERATOR rights and some require you to have SUPERVISOR rights. You have OPERATOR rights if the object by which you logged in is listed in (or is security equivalent to an object that is listed in) the FILE\_SERVER object's OPERATOR property.

A calling station has SUPERVISOR rights if the object by which the station logged in is the SUPERVISOR object or is security equivalent to the SUPERVISOR object.

If you do not have sufficient rights to call a given NCP, No Console Rights (0xC6) is returned.

Each server is listed in its own bindery as a bindery object of type FILE\_SERVER. The server object has an OPERATOR property that contains the object IDs of those objects that have OPERATOR rights (also referred to as console rights).

The NetWare FCONSOLE utility implements several of the protocols in this service.



# 40 NCPs

This section describes each of the Server Environment NCPs, their Request and Reply formats, and Return Values.

# Check Console Privileges 0x2222 23 200

Determines whether a station has console privileges on the target server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (200)	byte

## Return Values

If the station has console privileges, Successful is returned. Otherwise, No Console Rights is returned.

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights

# Clear Connection Number 0x2222 23 254

Logs out the specified station.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (254)	byte
10	ConnectionNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

Any resources or files that the station is using are released and the station's service connection is broken. The station must re-establish a service connection and log in before it can resume work on the file server.

If the calling station lacks supervisor privileges, No Console Rights is returned and the target station is not affected.

**Clear Connection Number** is equivalent to typing "CLEAR #" at the NetWare 2.x file server console or "CLEAR STATION #" at the NetWare 3.x or 4.x file server console, where # represents the connection number of the station to be cleared.

# Clear Connection Number (old) 0x2222 23 210

Logs out the specified station.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (210)	byte
10	ConnectionNumber	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

Any resources or files the station is using are released and the station's service connection is broken. The station must re-establish a service connection and log in before it can resume work on the file server.

If the calling station lacks supervisor privileges, No Console Rights is returned and the target station is not affected.

**Clear Connection Number** is equivalent to typing "CLEAR #" at the NetWare 2.x file server console or "CLEAR STATION #" at the NetWare 3.x or 4.x file server console, where # represents the connection number of the station to be cleared.

## See Also

**Clear Connection Number 0x2222 23 254 (page 859)**



# Disable File Server Login 0x2222 23 203

Disables future login requests.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (203)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

**Disable File Server Login** is usually called during some critical time, such as before taking the server down.

Use caution when calling this request. If, after making the request, you lose or destroy your service connection to the file server, you cannot create a new connection and cannot log in again. If no other user on the server has operator privileges, the server must be brought down from the console connected to the server and rebooted before any new users (including the operator) can access it.

If the calling station lacks operator privileges, No Console Rights is returned and the server's log state remains unchanged.

## See Also

**Enable File Server Login 0x2222 23 204 (page 864), Get File Server Login Status 0x2222 23 205 (page 907), Down File Server 0x2222 23 211 (page 863)**

# Disable Transaction Tracking 0x2222 23 207

Disables the transaction tracking functions on an SFT server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (207)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights

## Remarks

Usually, an operator will disable transaction tracking when a volume containing the transaction file is filled to capacity. To make space on the volume, the operator (or other users) will move or remove unnecessary files from the volume. You can re-enable transaction tracking by calling **Enable Transaction Tracking 0x2222 23 208**.

If the calling station does not have operator privileges, No Console Rights is returned and the server's transaction tracking status remains unchanged.

**Disable Transaction Tracking** has no effect on TTS in NetWare 4.11 or above.

## See Also

**Enable Transaction Tracking 0x2222 23 208 (page 865)**

# Down File Server 0x2222 23 211

Downs a file server from a remote console.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (211)	byte
10	ForceFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
255	0xFF	Failure

## Remarks

If *ForceFlag* is zero, the server determines whether any station have file server files open. If any files are in use or open, Failure is returned and the server does not go down. If no files are open or in use, the server shuts down and Successful is returned.

If *ForceFlag* is 0xFF, the server shuts down. All open files are automatically closed, and all unfinished transactions are automatically backed out.

If the calling station does not have operator privileges, No Console Rights is returned and the server is not brought down.

## See Also

**Disable File Server Login 0x2222 23 203 (page 861), Get File Server Login Status 0x2222 23 205 (page 907), Enable File Server Login 0x2222 23 204 (page 864)**

# Enable File Server Login 0x2222 23 204

Enables new login requests.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (204)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights

## Remarks

Clients that have already logged in will continue to receive service.

Enabling the server's log state also unlocks the Supervisor's account (if it has been locked because of intruder detection).

If the calling station does not have operator privileges, No Console Rights is returned and the file server's log state remains unchanged.

## See Also

**Disable File Server Login 0x2222 23 203 (page 861)**

# Enable Transaction Tracking 0x2222 23 208

Enables the transaction tracking functions of an SFT file server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (208)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights

## Remarks

TTS can be disabled manually by calling **Disable Transaction Tracking** or automatically by the server when the transaction volume becomes full. After freeing space on the transaction volume for the transaction files, the operator can call **Enable Transaction Tracking** to re-enable TTS.

If the calling station does not have operator privileges, No Console Rights is returned and the server’s transaction status remains unchanged.

**Enable Transaction Tracking** has no effect on TTS in NetWare 4.11 or above.

## See Also

**Disable Transaction Tracking 0x2222 23 207 (page 862)**

# Get Connection's Open Files 0x2222 23 235

Returns information (iteratively) about the files that are currently open on the specified connection.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (235)	byte
10	ConnectionNumber	word (Lo-Hi)
12	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfRecords	word (Lo-Hi)
...repeats NumberOfRecords times...		
12	TaskNumber	word (Lo-Hi)
14	LockType	byte
15	AccessControl	byte
16	LockFlag	byte
17	VolumeNumber	byte
18	DOSParentDirectoryEntry	long (Lo-Hi)
22	DOSDirectoryEntry	long (Lo-Hi)
26	ForkCount	byte
27	NameSpace	byte
28	NameLen	byte
29	FileName	byte[NameLen]

Offset	Content	Type
...		

## Parameters

### *TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

### *LockType*

(Reply) Specifies the bit flags indicating the file's lock status:

- 0 Locked
- 1 Open shareable
- 2 Logged
- 3 Open normal
- 6 TTS holding lock
- 7 Transaction flag set on this file

### *AccessControl*

Specifies the connection's access rights for the file:

- 0 Open for read by this client
- 1 Open for write by this client
- 2 Deny read requests from other stations
- 3 Deny write requests from other stations
- 4 File detached
- 5 TTS holding detach
- 6 TTS holding open

### *LockFlag*

Specifies the type of lock on the file:

- 0x00 Not locked
- 0xFE Locked by a file lock
- 0xFF Locked by Begin Share File Set

### *VolumeNumber*

Specifies the file's volume in a volume table on the file server.

### *DOSParentDirectoryEntry*

Specifies the file path for the parent to this directory (not a directory handle).

### *DOSDirectoryEntry*

Specifies the file path relative to this directory (not a directory handle).

### *FileName*

Specifies the NULL-terminated file name.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

*DOSParentDirectoryEntry* and *DOSDirectoryEntry* are DOS directory entries. The name space that is returned applies to *FileName* and does not imply that the directory entries are name space specific.

## See Also

**Get Connection's Task Information 0x2222 23 234 (page 876), Get Connection Using A File 0x2222 23 236 (page 880)**



# Get Connection's Open Files (old) 0x2222 23 219

Returns information (iteratively) about the files that are currently open on the specified connection.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (219)	byte
10	ConnectionNumber	word (Lo-Hi)
12	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfRecords	byte
...repeats NumberOfRecords times...		
11	TaskNumber	byte
12	LockType	byte
13	AccessControl	byte
14	LockFlag	byte
15	VolumeNumber	byte
16	DirectoryEntry	word
18	FileName	byte[14]
...		

## Parameters

### *TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

### *LockType*

(Reply) Specifies the bit flags indicating the file's lock status:

- 0 Locked
- 1 Open shareable
- 2 Logged
- 3 Open normal
- 6 TTS holding lock
- 7 Transaction flag set on this file

### *AccessControl*

Specifies the connection's access rights for the file:

- 0 Open for read by this client
- 1 Open for write by this client
- 2 Deny read requests from other stations
- 3 Deny write requests from other stations
- 4 File detached
- 5 TTS holding detach
- 6 TTS holding open

### *LockFlag*

Specifies the type of lock on the file:

- 0x00 Not locked
- 0xFE Locked by a file lock
- 0xFF Locked by Begin Share File Set

### *VolumeNumber*

Specifies the file's volume in a volume table on the file server.

### *DirectoryEntry*

Specifies the file path for the parent to this directory (not a directory handle).

### *FileName*

Specifies the NULL-terminated file name.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

Decimal	Hex	Description
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

**Get Connection's Open Files (old)** has been replaced by **Get Connection's Open Files** 0x2222 23 235.

## See Also

**Get Connection's Task Information** 0x2222 23 234 (page 876), **Get Connection Using A File** 0x2222 23 236 (page 880), **Get Connection's Open Files** 0x2222 23 235 (page 866), **Get Connection Using A File (old)** 0x2222 23 220 (page 884), **Get Connection Usage Statistics** 0x2222 23 229 (page 878), **Get File System Statistics** 0x2222 23 212 (page 912)

# Get Connection's Semaphores 0x2222 23 241

Returns a list of the semaphores that a connection is using.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (241)	byte
10	ConnectionNumber	word (Lo-Hi)
12	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfSemaphores	word (Lo-Hi)
... repeats NumberOfSemaphores times. ...		
12	OpenCount	word (Lo-Hi)
14	SemaphoreValue	word (Lo-Hi)
16	TaskNumber	word (Lo-Hi)
17	SemaphoreNameLen	byte
18	SemaphoreName	byte[SemaphroeNameLen]
...		

## Parameters

*OpenCount*

(Reply) Specifies the number of connections that have this semaphore open for use.

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore.

*TaskNumber*

Specifies the number of the connection's task that is using the semaphore.

*SemaphoreNameLen*

Specifies the length of the semaphore name.

*SemaphoreName*

Specifies the name of the semaphore.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

Because a connection might be using more semaphores than the server can report in one reply message, you might need to call **Get Connection's Semaphores** iteratively.

Set *LastRecordSeen* to zero on the first request. If a nonzero *NextRequestRecord* value is returned, the server has not reported all the information. For subsequent requests, set *LastRecordSeen* to the value previously returned in *NextRequestRecord* until *NextRequestRecord* is zero or an error is returned.

The server returns *NumberOfSemaphores*, which indicates that status of semaphores that the target connection is using.

A negative *SemaphoreValue* is usually interpreted as the number of workstations waiting for the service that is represented by the semaphore. A positive value indicates the number of free resources available in the resource pool governed by the semaphore.

## See Also

**Get Semaphore Information 0x2222 23 242 (page 942)**

# Get Connection's Semaphores (old) 0x2222 23 225

Returns a list of the semaphores that a connection is using.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (225)	byte
10	ConnectionNumber	word (Lo-Hi)
12	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfSemaphores	word (Lo-Hi)
... repeats NumberOfSemaphores times. ...		
12	OpenCount	word (Lo-Hi)
14	SemaphoreValue	word (Lo-Hi)
16	TaskNumber	word (Lo-Hi)
17	SemaphoreNameLen	byte
18	SemaphoreName	byte[SemaphroeNameLen]
...		

## Parameters

*OpenCount*

(Reply) Specifies the number of connections that have this semaphore open for use.

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore.

*TaskNumber*

(Reply) Specifies the number of the connection's task that is using the semaphore.

*SemaphoreNameLen*

(Reply) Specifies the length of the semaphore name.

*SemaphoreName*

(Reply) Specifies the name of the semaphore.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

Because a connection might be using more semaphores than the server can report in one reply message, you might need to call **Get Connection's Semaphores** iteratively.

Set *LastRecordSeen* to zero on the first request. If a nonzero *NextRequestRecord* value is returned, the server has not reported all the information. For subsequent requests, set *LastRecordSeen* to the value previously returned in *NextRequestRecord* until *NextRequestRecord* is zero or an error is returned.

The server returns *NumberOfSemaphores*, which indicates that status of semaphores that the target connection is using.

A negative *SemaphoreValue* is usually interpreted as the number of workstations waiting for the service that is represented by the semaphore. A positive value indicates the number of free resources available in the resource pool governed by the semaphore.

## See Also

**Get Connection's Semaphores 0x2222 23 241 (page 872)**

# Get Connection’s Task Information 0x2222 23 234

Returns control information about the current lock status of the connection’s active tasks.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (234)	byte
10	ConnectionNumber	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
9	AllAttrStruc	structure
??	NumberOfActiveTasks	byte
...repeats NumberOfActiveTasks times...		
??	TaskNumber	word (Lo-Hi)
??	TaskState	byte

## Parameters

*LockStatus*

(Reply) Specifies the status of the current lock:

- 0 Normal (connection free to run)
- 1 Connection waiting on a physical record lock
- 2 Connection waiting on a file lock
- 3 Connection waiting on a logical record lock
- 4 Connection waiting on a semaphore

*WaitRecord*

(Reply) Points to the WaitRecord structure.



### *NumberOfActiveTasks*

(Reply) Specifies the number of task and task state pairs to follow.

### *TaskNumber*

(Reply) Specifies the task number for which information is to be returned.

### *TaskState*

(Reply) Specifies the task's state at the time of the request:.

0x00 Normal task

0x01 TTS explicit transaction in progress

0x02 TTS implicit transaction in progress

0x04 Shared file set lock in progress

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

If the target connection is waiting for a lock request to be serviceable, *WaitRecord* contains information about the resource for which the connection is waiting. Its format depends on the value of *LockStatus*. If *LockStatus* is zero, no *WaitRecord* field exists in the reply message.

## See Also

**Get Connection Using A File 0x2222 23 236 (page 880), Get Connection's Open Files 0x2222 23 235 (page 866)**

# Get Connection Usage Statistics 0x2222 23 229

Returns the usage statistics for your own logical connection.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (3)	word (Hi-Lo)
9	SubFunctionCode (229)	byte
10	ConnectionNumber	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
12	BytesRead	byte[6]
18	BytesWritten	byte[6]
24	TotalRequestPackets	long

## Parameters

*SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

*BytesRead*

(Reply) Specifies the number of bytes the station has read on the specified connection since that connection was created.

*BytesWritten*

(Reply) Specifies the number of bytes that station has written on the specified connection since that connection was created.

# Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

# Remarks

Console operator rights are required to get usage statistics for connections other than yours.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

# See Also

**Get Connection Using A File 0x2222 23 236 (page 880), Get Connection's Open Files 0x2222 23 235 (page 866), Get Connection Using A File (old) 0x2222 23 220 (page 884), Get Connection's Open Files (old) 0x2222 23 219 (page 869), Get Connection's Task Information 0x2222 23 234 (page 876), Get Connection's Task Information 0x2222 23 234 (page 876), Get File System Statistics 0x2222 23 212 (page 912)**

# Get Connection Using A File 0x2222 23 236

Returns all logical connections using the specified file.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFunctionCode (236)	byte
10	DataStreamNumber	byte
11	VolumeNumber	byte
12	DirectoryBase	long (Lo-Hi)
16	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	UseCount	word (Lo-Hi)
12	OpenCount	word (Lo-Hi)
14	OpenForReadCount	word (Lo-Hi)
16	OpenForWriteCount	word (Lo-Hi)
18	DenyReadCount	word (Lo-Hi)
20	DenyWriteCount	word (Lo-Hi)
22	Locked	byte
23	ForkCount	byte
24	NumberOfRecords	word (Lo-Hi)
...repeats NumberOfRecords times...		
26	ConnectionNumber1	word (Lo-Hi)

Offset	Content	Type
28	TaskNumber1	word (Lo-Hi)
30	LockType1	byte
31	AccessControl1	byte
32	LockFlag1	byte

## Parameters

### *DataStreamNumber*

(Request) Specifies the data stream number if the name space is Macintosh:

- 0 Resource fork (or DOS)
- 1 Data fork

### *VolumeNumber*

(Request) Specifies the volume on which the file exists.

### *DirectoryBase*

(Request) Specifies the directory entry number—obtained from **File Search Initialize 0x2222 62 (page 324)** or **Convert Path to Dir Entry 0x2222 23 244 (page 301)**—that indicates the file or directory path.

### *LastRecordSeen*

(Request) Specifies the last record for which information is being returned.

### *NextRequestRecord*

(Reply) Specifies the value that *LastRecordSeen* must be set to for the next request.

### *UseCount*

(Reply) Specifies the number of tasks that have opened or logged the file.

### *OpenCount*

(Reply) Specifies the number of tasks that have the file open.

### *OpenForReadCount*

(Reply) Specifies the number of logical connections that have the file open for reading.

### *OpenForWriteCount*

(Reply) Specifies the number of logical connections that have the file open for writing.

### *DenyReadCount*

(Reply) Specifies the number of logical connections that have denied other connections the right to read from the file.

### *DenyWriteCount*

(Reply) Specifies the number of logical connections that have denied other connections that right to write to the file.

### *Locked*

(Reply) Specifies whether the file is locked exclusively:

- 0 Not locked exclusively
- 1 Locked exclusively

### *ConnectionNumber*

(Reply) Specifies the logical connection number of the workstation using the file.

### *TaskNumber1*

(Reply) Specifies the number of the task that is using the file.

### *LockType1*

(Reply) Specifies the file's lock status:

- 0 Locked
- 1 Open shareable
- 2 Logged
- 3 Open normal
- 6 TTS holding lock
- 7 Transaction flag set on this file

### *AccessControl1*

(Reply) Specifies the connection's access rights for the file:

- 0 Open for read by this client
- 1 Open for write by this client
- 2 Deny read requests from other stations
- 3 Deny write requests from other stations
- 4 File detached
- 5 TTS holding detach
- 6 TTS holding open

### *LockFlag1*

(Reply) Specifies the type of lock on the file:

- 0x00 Not locked
- 0xFE Locked by a file lock
- 0xFF Locked by Begin Share File Set

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

Repeat this request until *NextRequestRecord* is zero.

For the first request, set *LastRecordSeen* to zero. On subsequent requests, set *LastRecordSeen* to the value returned in *NextRequestRecord* on the previous request. If *NextRequestRecord* is zero, the file server has passed all information.

## See Also

**Get Connection's Open Files 0x2222 23 235 (page 866), Get Connection's Task Information 0x2222 23 234 (page 876)**

# Get Connection Using A File (old) 0x2222 23 220

Returns all logical connections using the specified file.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (220)	byte
10	LastRecordSeen	word (Hi-Lo)
12	DirectoryHandle	byte
13	PathLen	byte
14	FilePath	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	UseCount	word (Hi-Lo)
10	OpenCount	word (Hi-Lo)
12	OpenForReadCount	word (Hi-Lo)
14	OpenForWriteCount	word (Hi-Lo)
16	DenyReadCount	word (Hi-Lo)
18	DenyWriteCount	word (Hi-Lo)
20	NextRequestRecord	word (Hi-Lo)
22	Locked	byte
23	NumberOfRecords	word (Lo-Hi)
. . .repeats NumberOfRecords times. . .		
24	ConnectionNumber1	word (Hi-Lo)
26	TaskNumber1	byte



Offset	Content	Type
27	LockType1	byte
38	AccessControl1	byte
29	LockFlag1	byte

## Parameters

### *UseCount*

(Reply) Specifies the number of tasks that have opened or logged the file.

### *OpenCount*

(Reply) Specifies the number of tasks that have the file open.

### *OpenForReadCount*

(Reply) Specifies the number of logical connections that have the file open for reading.

### *OpenForWriteCount*

(Reply) Specifies the number of logical connections that have the file open for writing.

### *DenyReadCount*

(Reply) Specifies the number of logical connections that have denied other connections the right to read from the file.

### *DenyWriteCount*

(Reply) Specifies the number of logical connections that have denied other connections the right to write to the file.

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *Locked*

(Reply) Specifies whether the file is locked exclusively:

- 0 Not locked exclusively
- 1 Locked exclusively

### *ConnectionNumber1*

(Reply) Specifies the logical connection number of the workstation using the file.

### *TaskNumber1*

(Reply) Specifies the number of the task that is using the file.

### *LockType1*

(Reply) Specifies the file's lock status:

- 0 Locked
- 1 Open shareable
- 2 Logged
- 3 Open normal

- 6 TTS holding lock
- 7 Transaction flag set on this file

#### *AccessControl*

(Reply) Specifies the connection's access rights for the file:

- 0 Open for read by this client
- 1 Open for write by this client
- 2 Deny read requests from other stations
- 3 Deny write requests from other stations
- 4 File detached
- 5 TTS holding detach
- 6 TTS holding open

#### *LockFlag1*

(Reply) Specifies the type of lock on the file:

- 0x00 Not locked
- 0xFE Locked by a file lock
- 0xFF Locked by Begin Share File Set

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

Repeat this request until *NextRequestRecord* is zero.

For the first request, set *LastRecordSeen* to zero. On subsequent requests, set *LastRecordSeen* to the value returned in *NextRequestRecord* on the previous request. If *NextRequestRecord* is zero, the file server has passed all information.

## See Also

**Get Connection's Open Files 0x2222 23 235 (page 866), Get Connection's Task Information 0x2222 23 234 (page 876), Get Connection Using A File 0x2222 23 236 (page 880), Get Connection Usage Statistics 0x2222 23 229 (page 878), Get Connection's Open Files (old) 0x2222 23 219 (page 869), Get File System Statistics 0x2222 23 212 (page 912)**

# Get Disk Channel Statistics 0x2222 23 217

Returns the disk channel statistics for the specified disk channel number.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (217)	byte
10	DiskChannelNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long (Hi-Lo)
12	ChannelState	word (Hi-Lo)
14	ChannelSynchronizationState	word (Hi-Lo)
16	SoftwareDriverType	byte
17	SoftwareMajorVersionNumber	byte
18	SoftwareMinorVersionNumber	byte
19	SoftwareDescription	byte[65]
84	IOAddressesUsed	byte[4]
92	SharedMemoryAddresses	word[4] (Hi-Lo)
102	InterruptNumbersUsed	byte[4]
106	DMACHannelsUsed	byte[4]
110	FlagBits	byte
111	Reserved	byte
112	ConfigurationDescription	byte[80]

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *ChannelState*

(Reply) Specifies the state of the disk channel:

- 0x00 Channel is running
- 0x01 Channel is stopping
- 0x02 Channel is stopped
- 0x03 Channel is not functional

### *ChannelSynchronizationState*

(Reply) Specifies the control state of the disk channel:

- 0x00 Channel is not being used
- 0x02 NetWare is using the channel; no one else wants it
- 0x04 NetWare is using the channel; someone else wants it
- 0x06 Someone else is using the channel; NetWare does not need it
- 0x08 Someone else is using the channel; NetWare needs it
- 0x0A Someone else has released the channel; NetWare should use it

### *SoftwareDriverType*

(Reply) Specifies the type of disk driver software that is installed in the disk channel.

### *SoftwareMajorVersionNumber*

(Reply) Specifies the major version of the disk driver software that is installed on the disk channel.

### *SoftwareMinorVersionNumber*

(Reply) Specifies the minor version of the disk driver software that is installed on the disk channel.

### *SoftwareDescription*

(Reply) Specifies the NULL-terminated string describing the disk driver software.

### *IOAddressesUsed*

(Reply) Specifies the two addresses the disk driver uses to control the disk channel.

### *SharedMemoryAddresses*

(Reply) Specifies the two shared memory addresses (offsets).

### *InterruptedNumbersUsed*

(Reply) Specifies the two interrupt numbers that the disk driver uses to communicate with the disk channel.

### *DMAChannelIsUsed*

(Reply) Specifies the DMA controllers used by the disk driver to control the disk channel.

### *ConfigurationDescription*

(Reply) Specifies the NULL-terminated string that contains the channel's current IO driver configuration.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

You must have OPERATOR rights in order to make this request.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second.) When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

## See Also

**Get Disk Utilization 0x2222 23 14 (page 890), Get File System Statistics 0x2222 23 212 (page 912)**

# Get Disk Utilization 0x2222 23 14

Returns how much physical space the specified trustee ID is using on the given volume.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (6)	word (Hi-Lo)
9	SubFunctionCode (14)	byte
10	VolumeNumber	byte
11	TrusteeID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	VolumeNumber	byte
9	TrusteeID	long (Hi-Lo)
13	DirectoryCount	word (Hi-Lo)
15	FileCount	word (Hi-Lo)
17	ClusterCount	word (Hi-Lo)

## Parameters

### *DirectoryCount*

(Reply) Specifies the number of directories on the volume that are owned by the trustee ID.

### *FileCount*

(Reply) Specifies the number of files on the volume that are owned by the trustee ID.

### *ClusterCount*

(Reply) Specifies the number of physical volume clusters that the trustee's files occupy on the volume.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
161	0xA1	Directory I/O Error
242	0xF2	No Object Read

## Remarks

Clients who are supervisor equivalent can make this request for any object. Clients that do not have supervisor rights can make this request for the object used when logging in.

To map a cluster count to a count of the number of bytes used, call **Get Volume Info With Number** 0x2222 18.

# Get Drive Mapping Table 0x2222 23 215

Returns the file server's drive mapping table.

NetWare Servers: 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (215)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long (Hi-Lo)
12	SFTSupportLevel	byte
13	LogicalDriveCount	byte
14	PhysicalDriveCount	byte
15	DiskChannelTable	byte[5]
20	PendingIOCommands	word (Hi-Lo)
22	DriveMappingTable	byte[32]
54	DriveMirrorTable	byte[32]
86	DeadMirrorTable	byte[32]
118	ReMirrorDriveNumber	byte
119	Filler	byte
120	ReMirrorCurrentOffset	long (Hi-Lo)
124	SFTErrorTable	word[60] (Hi-Lo)

## Parameters

*SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.



### *SFTSupportLevel*

(Reply) Specifies the SFT level offered by the file server:

- 1 File server offers hot disk error fixing
- 2 File server offers disk mirroring and transaction tracking
- 3 File server offers physical file server mirroring

### *LogicalDriveCount*

(Reply) Specifies the number of logical drives attached to the server.

### *PhysicalDriveCount*

(Reply) Specifies the number of physical disk units that are attached to the server.

### *DiskChannelTable*

(Reply) Specifies the disk channels that exist on the server and their drive types (5-byte table):

- 1 XT
- 2 AT
- 3 SCSI
- 4 disk coprocessor
- 50-255 Value Added Disk Drive (VADD)

### *PendingIOCommands*

(Reply) Specifies the number of outstanding disk controller commands.

### *DriveMappingTable*

(Reply) Specifies the primary physical drive to which each logical drive is mapped (32-byte table).

### *DriveMirrorTable*

(Reply) Specifies the secondary physical drive to which each logical drive is mapped (32-byte table).

### *DeadMirrorTable*

(Reply) Specifies the secondary physical drive to which each logical drive was mapped.

### *ReMirrorDriveNumber*

(Reply) Specifies the physical drive number of the disk that is currently being remirrored.

### *Filler*

(Reply) Specifies no information.

### *ReMirrorCurrentOffset*

(Reply) Specifies the block number that is currently being remirrored.

### *SFTErrorTable*

(Reply) Specifies the SFT internal error counters (60-byte table).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

If the calling station does not have operator privileges, No Console Rights is returned.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. When this field reaches 0xFFFFFFFF, it wraps back to zero.

If the file server supports SFT Level II or above and disks are mirrored, *LogicalDriveCount* will be lower than the actual number of physical disk subsystems that are attached to the file server. The file server's OS considers mirrored disks to be one logical drive.

If *DriveMappingTable*, *DriveMirrorTable*, and *DeadMirrorTable* are 0xFF, no such logical drive exists.

If the entry in *DriveMappingTable* shows that a drive is not currently mirrored, the table can still be used to determine which drive previously mirrored the logical drive. *DeadMirrorTable* is used to remirror a logical drive after a mirror failure.

If *ReMirrorDriveNumber* is 0xFF, no disk is currently being remirrored.

## See Also

**Get Disk Channel Statistics 0x2222 23 217 (page 887), Get File System Statistics 0x2222 23 212 (page 912)**

# Get File Server Date And Time 0x2222 20

Returns the current data and time being kept by the file server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (20)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	Year	byte
9	Month	byte
10	Day	byte
11	Hour	byte
12	Minute	byte
13	Second	byte
14	DayOfWeek	byte

## Parameters

*Year*

(Reply) Specifies the year; values from 80 to 99 correspond to the years 1980 through 1999, values from 100 to 179 correspond to the years 2000 through 2079.

*Month*

(Reply) Specifies the month, from 1 to 12 corresponding to January through December.

*Day*

(Reply) Specifies the day, from 1 to 31.

*Hour*

(Reply) Specifies the hour, from 0 to 23 with zero indicating midnight and 23 indicating 11 p.m.

### *Minute*

(Reply) Specifies the minutes, from 0 to 59.

### *Second*

(Reply) Specifies the second, from 0 to 59.

### *DayOfWeek*

(Reply) Specifies the day of the week, from 0 to 6 (zero = Sunday and 6 = Saturday).

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Any client can make this request.

The system time clock is a 7-byte clock.

## See Also

**Set File Server Date And Time 0x2222 23 202 (page 964)**

# Get File Server Description Strings 0x2222 23 201

Returns up to 512 bytes of information (in descriptive strings) about the server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (201)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	DescriptionStrings	byte[512]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

## Remarks

Each returned string is NULL-terminated and can include information such as a company’s name, the NetWare version, the revision date, a copyright notice, etc.

Any client can make this request.

## See Also

**Get File Server Information 0x2222 23 17 (page 898)**

# Get File Server Information 0x2222 23 17

Returns information about a server to which you have a service connection.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (17)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	ServerName	byte[48]
56	FileServiceVersion	byte
57	FileServiceSubVersion	byte
58	MaximumServiceConnections	word (Hi-Lo)
60	ConnectionsInUse	word (Hi-Lo)
62	NumberMountedVolumes	word (Hi-Lo)
64	Revision	byte
65	SFTLevel	byte
66	TTSLevel	byte
67	MaxConnectionsEverUsed	word (Hi-Lo)
69	AccountVersion	byte
70	VAPVersion	byte
71	QueueVersion	byte
72	PrintVersion	byte
73	VirtualConsoleVersion	byte
74	RestrictionLevel	byte

Offset	Content	Type
75	InternetBridge	byte
76	MixedModePathFlag	byte
77	LocalLoginInfoCcode	byte
78	ProductMajorVersion	word (Hi-Lo)
80	ProductMinorVersion	word (Hi-Lo)
82	ProductRevisionVersion	word (Hi-Lo)
84	OSLanguageID	byte
85	64BitOffsetsSupportedFlag	byte
86	Reserved	byte[50]

## Parameters

### *ServerName*

(Reply) Specifies the server name.

### *FileServiceVersion*

(Reply) Specifies the major NetWare version number.

### *FileServiceSubversion*

(Reply) Specifies the minor NetWare version number.

### *MaximumServiceConnections*

(Reply) Specifies the maximum number of connection slots that the server has allocated since it has been up.

### *ConnectionsInUse*

(Reply) Specifies the number of connections that are currently using the server.

### *NumberMountedVolumes*

(Reply) Specifies the maximum number of volumes that the server supports.

### *Revision*

(Reply) Specifies the revision level of the NetWare version.

### *SFTLevel*

(Reply) Specifies the SFT level that the server OS is using.

### *TTSLevel*

(Reply) Specifies the TTS level that the server OS is using.

### *MaxConnectionsEverUsed*

(Reply) Specifies the maximum number of licensed connections in use at one time.

*AccountVersion*

(Reply) Specifies the accounting version number.

*VAPVersion*

(Reply) Specifies the VAP version number.

*QueueVersion*

(Reply) Specifies the queueing version number.

*PrintVersion*

(Reply) Specifies the print server version number.

*VirtualConsoleVersion*

(Reply) Specifies the virtual console version number.

*RestrictionLevel*

(Reply) Specifies the security restriction version number.

*InternetBridge*

(Reply) Specifies the Internet bridge support version number.

*MixedModePathFlag*

(Reply) Specifies whether mixed mode path handling is available:

0 Mixed mode path handling is not available

1 Mixed mode path handling is available

*LocalLoginInfoCcode*

(Reply) Specifies the completion code returned from the GetLocalLoginInfo function.

*ProductMajorVersion*

(Reply) Specifies the product's major version.

*ProductMinorVersion*

(Reply) Specifies the product's minor version.

*ProductRevisionVersion*

(Reply) Specifies the product's revision version.

*OSLanguageID*

(Reply) Specifies the language number for the language that is currently enabled on the server.

*64BitOffsetsSupportedFlag*

(Reply) Specifies whether the server supports reading from, writing to, and locking offsets greater than 4GB.

*Reserved*

(Reply) Is reserved for future use.



## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

## Remarks

Any client can make this request without first calling Login.

*MaximumServiceConnections* is not an indication of how many connections the server supports. The connections that increment this value might or might not be licensed.

## See Also

**[Get File Server Description Strings 0x2222 23 201 \(page 897\)](#)**

# Get File Server LAN I/O Statistics 0x2222 23 231

Returns statistical information about packets being received and sent by a file server.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (231)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long
12	ConfiguredMaxRoutingBuffers	word
14	ActualMaxUsedRoutingBuffers	word
16	CurrentlyUsedRoutingBuffers	word
18	TotalFileServicePackets	long
22	TurboUsedForFileService	word
24	PacketsFromInvalidConnection	word
26	BadLogicalConnectionCount	word
28	PacketsReceivedDuringProcessing	word
30	RequestsReprocessed	word
32	PacketsWithBadSequenceNumber	word
34	DuplicateRepliesSent	word
36	PositiveAcknowledgesSent	word
38	PacketsWithBadRequestType	word
40	AttachDuringProcessing	word
42	AttachWhileProcessingAttach	word

Offset	Content	Type
44	ForgedDetachedRequests	word
46	DetachForBadConnectionNumber	word
48	DetachDuringProcessing	word
50	RepliesCancelled	word
52	PacketsDiscardedByHopCount	word
54	PacketsDiscardedUnknownNet	word
56	IncomingPacketDiscardedNoDGroup	word
58	OutgoingPacketDiscardedNoTurboBuffer	word
60	IPXNotMyNetwork	word
62	NetBIOSBroadcastWasPropogated	long
66	TotalOtherPackets	long
70	TotalRoutedPackets	long

## Parameters

### *SystemIntervalMarked*

(Reply) Specifies how long the file server has been up.

### *ConfiguredMaxRoutingBuffers*

(Reply) Specifies the number of routing buffers the network is configured to handle.

### *ActualMaxUsedRoutingBuffers*

(Reply) Specifies the maximum number of routing buffers that have been in use simultaneously since the server was brought up.

### *CurrentlyUsedRoutingBuffers*

(Reply) Specifies the number of routing buffers that are being used by the server.

### *TotalFileServicePackets*

(Reply) Specifies the number of request packets serviced by the file server.

### *TurboUsedForFileService*

(Reply) Specifies the number of times that file service request packets were stored in routing buffers.

### *PacketsFromInvalidConnection*

(Reply) Specifies the count of all request packets with invalid logical connection numbers.

### *PacketsReceivedDuringProcessing*

(Reply) Specifies the number of times a new request is received while the previous request is still being processed.

*RequestsReprocessed*

(Reply) Specifies the count of requests reprocessed by the server.

*PacketsWithBadSequenceNumber*

(Reply) Specifies the count of request packets that the server received from a connection whose packet sequence number does not match the current sequence number or the next sequence number.

*DuplicateRepliesSent*

(Reply) Specifies the number of requests packets for which the server had to send a duplicate reply.

*PositiveAcknowledgesSent*

(Reply) Specifies the number of times that a client repeats a request that is being serviced.

*PacketsWithBadRequestType*

(Reply) Specifies the number of request packets that contained an invalid request type.

*AttachDuringProcessing*

(Reply) Specifies the number of times the server is requested to create a service connection by clients for which the server is currently processing a request.

*AttachWhileProcessingAttach*

(Reply) Specifies the number of times the files server receives a request to create a service connection while still servicing the same request that was received previously.

*ForgedDetachedRequests*

(Reply) Specifies the number of requests that were received to terminate a connection whose source address does not match the address the server has assigned to the connection.

*DetachForBadConnectionNumber*

(Reply) Specifies the number of times a request was received to terminate a connection for a connection number that is not supported by the server.

*DetachDuringProcessing*

(Reply) Specifies the number of requests that were received to terminate a connection while requests are still being processed for the client.

*RepliesCancelled*

(Reply) Specifies the number of replies that were cancelled because the connection was reallocated during processing.

*PacketsDiscardedByHopCount*

(Reply) Specifies the number of packets discarded because they have passed through more than 16 bridges without reaching their destination.

*PacketsDiscardedUnknownNet*

(Reply) Specifies the number of packets that were discarded because their destination network is unknown to the server.

### *IncomingPacketDiscardedNoDGroup*

(Reply) Specifies the number of incoming packets that were received in a routing buffer that needed to be transferred to a DGroup buffer so that the socket dispatcher could transfer the packet to the correct socket.

### *OutgoingPacketDiscardedNoTurboBuffer*

(Reply) Specifies the number of packets the server attempted to send that were lost because no routing buffers were available.

### *IPXNotMyNetwork*

(Reply) Specifies the count of received packets that were destined for the B or C side drivers.

### *NetBIOSBroadcastWasPropagated*

(Reply) Specifies the count of NetBIOS packets circulated through this network.

### *TotalOtherPackets*

(Reply) Specifies the count of all received packets that were not requests for file services.

### *TotalRoutedPackets*

(Reply) Specifies the count of all packets routed by the server.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

## Remarks

Any client can make this request without first calling Login.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. When this value reaches 0xFFFFFFFF, it wraps back to zero.

If a packet's logical connection number has not been allocated or the packet's source address doesn't match the address assigned by the file server, the connection number is invalid.

*PacketsReceivedDuringProcessing* counts packets that are received when the client generates a duplicate request while the response to the first request is being sent to the client. The file server reprocesses these requests unnecessarily.

Requests can be reprocessed if the client repeats a request for one that did not receive a response.

Packets with bad sequence numbers are discarded.

Duplicate replies are sent only for requests the server cannot process.

If the server is requested to create a service connection by clients for which the server is currently processing a request, the server does not respond to the request currently being processed and recreates a connection with the client (station).

The maximum number of bridges depends on the particular implementation of this request. For NetWare 2.x, the maximum number of bridges is 16.

If no buffers are available for DGroup packets that need to be transferred, the packet is lost.

## See Also

**[Get LAN Driver Configuration Information 0x2222 23 227 \(page 915\)](#)**

# Get File Server Login Status 0x2222 23 205

Returns whether user logins are currently allowed on the target server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (205)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	UserLoginAllowed	long

## Parameters

*UserLoginAllowed*

(Reply) Specifies if user logins are currently allowed:

- 0 Client cannot log in
- nonzero Clients can log in

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
251	0xFB	Unknown Request

## Remarks

You do not need console privileges to call **Get File Server Login Status**.

## See Also

**Disable File Server Login 0x2222 23 203 (page 861), Down File Server 0x2222 23 211 (page 863)**



# Get File Server Misc Information 0x2222 23 232

Returns miscellaneous information about the file server.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (232)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long
12	ProcessorType	byte
13	Reserved	byte
14	NumberOfServiceProcesses	byte
15	ServerUtilizationPercentage	byte
16	ConfiguredMaxBinderyObjects	word
18	ActualMaxBinderyObjects	word
20	CurrentUsedBinderyObjects	word
22	TotalServerMemory	word (Hi-Lo)
24	WastedServerMemory	word (Hi-Lo)
26	NumberOfDynamicMemoryAreas (1...3)	word (Hi-Lo)
... repeats NumerOfDynamicMemoryAreas times. ...		
28	TotalDynamicSpace	long (Hi-Lo)
32	MaxUsedDynamicSpace	long
36	CurrentUsedDynamicSpace	long

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *ProcessorType*

(Reply) Specifies the processor type:

- 0 Motorola 68000
- 1 Intel 8088 or 8086
- 2 80286

### *NumberOfServiceProcesses*

(Reply) Specifies the number of processes in the server that handle incoming service requests.

### *ServerUtilizationPercentage*

(Reply) Specifies the current server utilization percentage (0...100).

### *ConfiguredMaxBinderyObjects*

(Reply) Specifies the maximum number of bindery objects that the file server will track (0 = unlimited).

### *ActualMaxBinderyObjects*

(Reply) Specifies the maximum number of bindery objects that have been used concurrently since the file server was brought up.

### *CurrentUsedBinderyObjects*

(Reply) Specifies the number of bindery objects currently in use on the server.

### *TotalServerMemory*

(Reply) Specifies the total amount of memory installed on the server.

### *WastedServerMemory*

(Reply) Specifies the amount of memory that the server has determined it is not using.

### *NumberOfDynamicMemoryAreas*

(Reply) Specifies the number of dynamic memory areas (1-3).

### *TotalDynamicSpace*

(Reply) Specifies the total amount of memory in the dynamic memory area.

### *MaxUsedDynamicSpace*

(Reply) Specifies the amount of memory in the dynamic memory area that has been in use since the server was brought up.

### *CurrentUsedDynamicSpace*

(Reply) Specifies the amount of memory in the dynamic memory area that is currently in use.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

*ServerUtilizationPercentage* is updated once per second.

If disk resource limitation is not installed on the file server, *ConfiguredMaxBinderyObjects* is zero and an unlimited number of bindery objects can be created. Otherwise, *ConfiguredMaxBinderyObjects* contains that number that was configured during installation. The file server will not allow the number of bindery objects to exceed this value.

*ActualMaxBinderyObjects* and *CurrentUsedBinderyObjects* have no meaning if *ConfiguredMaxBinderyObjects* is zero.

# Get File System Statistics 0x2222 23 212

Returns statistics about a server's file system.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (212)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long (Hi-Lo)
12	ConfiguredMaxOpenFiles	word
14	ActualMaxOpenFiles	word
16	CurrentOpenFiles	word
18	TotalFilesOpened	long
22	TotalReadRequests	long
26	TotalWriteRequests	long
30	CurrentChangedFATs	word
32	TotalChangedFATs	long
36	FATWriteErrors	word
38	FatalFATWriteErrors	word
40	FATScanErrors	word
42	ActualMaxIndexedFiles	word
44	ActiveIndexedFiles	word
46	AttachedIndexedFiles	word
48	AvailableIndexedFiles	word

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *ConfiguredMaxOpenFiles*

(Reply) Specifies the number of files the server can open simultaneously.

### *ActualMaxOpenFiles*

(Reply) Specifies the number of file open simultaneously since the server was brought up.

### *CurrentOpenFiles*

(Reply) Specifies the number of files the server has open.

### *TotalFilesOpened*

(Reply) Specifies the number of files the server has opened since the server was brought up.

### *TotalReadRequests*

(Reply) Specifies the number of read requests the server has received since it was brought up.

### *TotalWriteRequests*

(Reply) Specifies the number of write requests the server has received since it was brought up.

### *CurrentChangedFATs*

(Reply) Specifies the number of current FAT sectors that the file system has modified.

### *TotalChangedFATs*

(Reply) Specifies the number of FAT sectors that the file system has modified since the server was brought up.

### *FATWriteErrors*

(Reply) Specifies the number of disk write errors that have occurred in writing FAT sectors to the disk.

### *FatalFATWriteErrors*

(Reply) Specifies the number of disk write errors that occurred in both the original and duplicate FAT sector.

### *FATScanErrors*

(Reply) Specifies the number of times an internally inconsistent state existed in the file system.

### *ActualMaxIndexedFiles*

(Reply) Specifies the number of indexed files that were simultaneously active in the server since it was brought up.

### *ActiveIndexedFiles*

(Reply) Specifies the count of files that are currently active, open, and indexed.

### *AttachedIndexedFiles*

(Reply) Specifies the count of indexed files that are ready for indexing but are not ready for use.

### *AvailableIndexedFiles*

(Reply) Specifies the count of file indexes that are available for use.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

**Get File System Statistics** can be called repeatedly to return updated information.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

*CurrentOpenFiles* exceeds the previously recorded *ActualMaxOpenFiles* count because it reflects both files that clients have open and any internal files-such as the bindery-that the server has open.

If *TotalFilesOpened*, *TotalReadRequests*, and *TotalWriteRequests* reach 0xFFFFFFFF, they wrap back to zero.

FAT sectors are modified when files are extended or truncated. All FAT sectors are duplicated on the disk, ensuring that a single disk failure will not result in a loss of all the FAT tables. If a FAT sector is lost, its duplicate is automatically used. When a disk write error occurs in both the original and the duplicate FAT sector, a fatal error occurs. Fatal FAT write errors occur because the file system cannot recover the information required to determine where a given file resides on a volume. However, since a copy of the FAT table is kept in memory, the file server continues to function.

Attached indexed files are not currently open, but they have indexes built in memory. These indexes are reused if the attached file is reopened, or rebuilt for other files (if needed).

## See Also

**Get Connection Using A File 0x2222 23 236 (page 880), Get Connection Using A File (old) 0x2222 23 220 (page 884), Get Connection Usage Statistics 0x2222 23 229 (page 878), Get Connection's Open Files 0x2222 23 235 (page 866), Get Connection's Task Information 0x2222 23 234 (page 876)**

# Get LAN Driver Configuration Information 0x2222 23 227

Returns configuration information for the LAN drivers that installed on the file server.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (227)	byte
10	LANDriverNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	NetworkAddress	byte[4]
12	HostAddress	byte[6]
18	BoardInstalled	byte
19	OptionNumber	byte
20	ConfigurationText	byte[160]

## Parameters

*NetworkAddress*

(Reply) Specifies the LAN board number to which the driver is attached.

*HostAddress*

(Reply) Specifies the host address of the LAN board that the driver is controlling.

*BoardInstalled*

(Reply) Specifies whether a drive is installed in the specified LAN board.

*OptionNumber*

(Reply) Specifies the option selected for the driver during configuration; the option specifies the hardware setup the driver uses when running the LAN board.

### *ConfigurationText*

(Reply) Specifies one or more NULL-terminated strings that contain configuration information such as the LAN board type ("NetWare Ethernet NP600") and the LAN board hardware settings ("IRQ=9, IOAddress = 300h, DMA = 7").

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
255	0xFF	Failure

## Remarks

If *BoardInstalled* is zero, all returned fields are undefined.

## See Also

**Get File Server LAN I/O Statistics 0x2222 23 231 (page 902)**



# Get Logical Record Information 0x2222 23 240

Returns information about a logical record.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + LogicalRecordNameLen)	word (Hi-Lo)
9	SubFunctionCode (240)	byte
10	LastRecordSeen	word (Lo-Hi)
12	LogicalRecordNameLen	byte
13	LogicalRecordName	byte[LogicalRecordNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	ShareableLockCount	word (Lo-Hi)
10	UseCount	word (Lo-Hi)
12	Locked	byte
13	NextRequestRecord	word (Lo-Hi)
15	NumberOfRecords	word (Lo-Hi)
...repeats NumberOfRecords times...		
17	ConnectionNumber	word (Lo-Hi)
19	TaskNumber	word (Lo-Hi)
21	LockStatus	byte

## Parameters

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

### *LogicalRecordNameLen*

(Request) Specifies the length of *LogicalRecordName*.

### *LogicalRecordName*

(Request) Specifies the name of the logical record.

### *ShareableLockCount*

(Reply) Specifies the number of logical connections that have a shareable lock on the logical record.

### *UseCount*

(Reply) Specifies the number of logical connections that have the logical record logged.

### *Locked*

(Reply) Specifies whether the logical record is exclusively locked:

- 0 Not exclusively locked
- 1 Exclusively locked

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *NumberOfRecords*

(Reply) Specifies the number of records in the reply.

### *ConnectionNumber*

(Reply) Specifies the logical connection that is using the logical record.

### *TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

### *LockStatus*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
255	0xFF	Failure

## Remarks

You must have console operator privileges to call **Get Logical Record Information**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**[Get Logical Records By Connection 0x2222 23 239 \(page 923\)](#)**

# Get Logical Record Information (old) 0x2222 23 224

Returns information about a logical record.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + LogicalRecordNameLen)	word (Hi-Lo)
9	SubFunctionCode (224)	byte
10	LastRecordSeen	word (Lo-Hi)
12	LogicalRecordNameLen	byte
13	LogicalRecordName	byte[LogicalRecordNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	UseCount	word (Hi-Lo)
10	ShareableLockCount	word (Hi-Lo)
12	NextRequestRecord	word
14	Locked	byte
15	NumberOfRecords	byte
...repeats NumberOfRecords times...		
16	ConnectionNumber	word (Hi-Lo)
18	TaskNumber	byte
19	LockStatus	byte

## Parameters

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

*LogicalRecordNameLen*

(Request) Specifies the length of *LogicalRecordName*.

*LogicalRecordName*

(Request) Specifies the name of the logical record.

*UseCount*

(Reply) Specifies the number of logical connections that have the logical record logged.

*ShareableLockCount*

(Reply) Specifies the number of logical connections that have a shareable lock on the logical record.

*NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

*Locked*

(Reply) Specifies whether the logical record is exclusively locked:

- 0 Not exclusively locked
- 1 Exclusively locked

*NumberOfRecords*

(Reply) Specifies the number of records in the reply.

*ConnectionNumber*

(Reply) Specifies the logical connection that is using the logical record.

*TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

*LockStatus*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

You must have console operator privileges to call **Get Logical Record Information (old)**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Logical Records By Connection 0x2222 23 239 (page 923), Get Logical Record Information 0x2222 23 240 (page 917), Get Logical Records By Connection (old) 0x2222 23 223 (page 925)**

# Get Logical Records By Connection 0x2222 23 239

Returns the logical records that a connection has logged with the file server.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (239)	byte
10	TargetConnectionNumber	word (Lo-Hi)
12	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfRecords	word (Lo-Hi)
... repeats NumberOfRecords times. ...		
12	TaskNumber	word (Lo-Hi)
14	LockStatus	byte
15	LockNameLen	byte
16	LockName	byte[LockNameLen]

## Parameters

*TargetConnectionNumber*

(Request) Specifies the logical connection that has the record exclusively locked.

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

*NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

*NumberOfRecords*

(Reply) Specifies the number of records in the reply.

*LogicalRecordName*

(Request) Specifies the name of the logical record.

*TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

*LockStatus*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

*LockNameLen*

(Reply) Specifies the length of *LockName*.

*LockName*

(Reply) Specifies the name of the logical lock.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

You must have console operator privileges to call **Get Logical Records By Connection**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Logical Record Information 0x2222 23 240 (page 917)**



# Get Logical Records By Connection (old) 0x2222 23 223

Returns the logical records that a connection has logged with the file server.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (223)	byte
10	TargetConnectionNumber	word
12	LastRecordSeen	word (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word
10	NumberOfRecords	byte
...repeats NumberOfRecords times...		
11	TaskNumber	byte
12	LockStatus	byte
13	LockNameLen	byte
14	LockName	byte[LockNameLen]

## Parameters

*TargetConnectionNumber*

(Request) Specifies the logical connection that has the record exclusively locked.

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

*NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

*NumberOfRecords*

(Reply) Specifies the number of records in the reply.

*LogicalRecordName*

(Request) Specifies the name of the logical record.

*TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

*LockStatus*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

*LockNameLen*

(Reply) Specifies the length of *LockName*.

*LockName*

(Reply) Specifies the name of the logical lock.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

You must have console operator privileges to call **Get Logical Records By Connection**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Logical Record Information 0x2222 23 240 (page 917), Get Logical Record Information (old) 0x2222 23 224 (page 920), Get Logical Records By Connection 0x2222 23 239 (page 923)**

# Get Network Serial Number 0x2222 23 18

Returns a file server's serial and application numbers.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (18)	byte

## Reply Format

Offset	Content	Type
Reply header		
9	ServerSerialNumber	long (Lo-Hi)
13	ApplicationNumber	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

## Remarks

The combination of the server's serial number and the application number is unique.

# Get Object's Remaining Disk Space 0x2222 23 230

Returns the number of unused disk blocks available to the specified object.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (230)	byte
10	ObjectID	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long (Hi-Lo)
12	ObjectID	long (Lo-Hi)
16	UnusedDiskBlocks	long (Hi-Lo)
20	RestrictionsEnforced	byte

## Parameters

### *ObjectID*

(Request) Specifies the object ID for the object that is requesting information.

### *SystemIntervalMarker*

(Reply) Specifies the length of time that the file server has been up.

### *UnusedDiskBlocks*

(Reply) Specifies the number of blocks the file server has that are available to allocate to a bindery object.

### *RestrictionsEnforced*

(Reply) Specifies whether the file server OS can place limitations on disk resources:

0x00 Enforced

0xFF Not enforced

# Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

# Remarks

You can obtain information for only the object ID by which you are logged in. You must have console operator privileges to call **Get Object's Remaining Disk Space** for any other object ID.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

# Get Physical Record Locks By Connection And File 0x2222 23 237

Returns a logical connection's physical record locks within a file.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (22)	word (Hi-Lo)
9	SubFunctionCode (237)	byte
10	TargetConnectionNumber	word (Lo-Hi)
12	DataStreamNumber	byte
13	VolumeNumber	byte
14	DirectoryBase	long (Lo-Hi)
18	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfLocks	word (Lo-Hi)
...repeats NumberOfLocks times...		
12	TaskNumber	word (Lo-Hi)
14	LockType	byte
15	RecordStart	long (Lo-Hi)
19	RecordEnd	long (Lo-Hi)

## Parameters

*TargetConnectionNumber*

(Request) Specifies the logical connection number of the client that is using the file.

### *DataStreamNumber*

(Request) Specifies the data stream number for the Macintosh name space:

- 0 Resource fork (or DOS)
- 1 Data fork

### *VolumeNumber*

(Request) Specifies the volume on which the file exists.

### *DirectoryBase*

(Request) Specifies the directory entry number for the file or directory path (obtained from calling **File Search Initialize** 0x2222 62).

### *LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *NumberOfLocks*

(Reply) Specifies the number of physical record locks.

### *TaskNumber*

(Reply) Specifies the task number within the workstation that has the file open.

### *LockType*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

### *RecordStart*

(Reply) Specifies the byte offset where the record begins within the file.

### *RecordEnd*

(Reply) Specifies the byte offset where the record ends within the file.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

Decimal	Hex	Description
255	0xFF	No Files Found

## Remarks

You must have console operator privileges to call **Get Physical Record Locks By Connection And File**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Physical Record Locks By File 0x2222 23 238 (page 936)**



# Get Physical Record Locks By Connection And File (old) 0x2222 23 221

Returns a logical connection's physical record locks within a file.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (22)	word (Hi-Lo)
9	SubFunctionCode (221)	byte
10	TargetConnectionNumber	word
12	LastRecordSeen	word
14	VolumeNumber	byte
15	DirectoryID	word
17	FileName	byte[14]

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word
10	NumberOfLocks	byte
11	Reserved	byte
...repeats NumberOfLocks times...		
12	TaskNumber1	byte
13	LockType1	byte
14	RecordStart1	long (Hi-Lo)
18	RecordEnd1	long (Hi-Lo)

## Parameters

### *TargetConnectionNumber*

(Request) Specifies the logical connection number of the client that is using the file.

### *LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

### *VolumeNumber*

(Request) Specifies the volume on which the file exists.

### *DirectoryID*

(Request) Specifies the file path relative to this directory (obtained from calling **File Search Initialize** 0x2222 62).

### *FileName*

(Request) Specifies the name of the file for which information is being requested.

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *NumberOfLocks*

(Reply) Specifies the number of physical record locks.

### *TaskNumber1*

(Reply) Specifies the task number within the workstation that has the file open.

### *LockType1*

(Reply) Specifies the bits indicating the file's lock status:

- 0 Locked exclusive
- 1 Locked shareable
- 2 Logged
- 6 Lock is held by TTS

### *RecordStart1*

(Reply) Specifies the byte offset where the record begins within the file.

### *RecordEnd1*

(Reply) Specifies the byte offset where the record ends within the file.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

Decimal	Hex	Description
253	0xFD	Bad Station Number
255	0xFF	No Files Found

## Remarks

You must have console operator privileges to call **Get Physical Record Locks By Connection And File (old)**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Physical Record Locks By File 0x2222 23 238 (page 936), Get Physical Record Locks By Connection And File 0x2222 23 237 (page 930), Get Physical Record Locks By File (old) 0x2222 23 222 (page 939)**

# Get Physical Record Locks By File 0x2222 23 238

Returns a logical connection's physical record locks within a file.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (238)	byte
10	DataStreamNumber (fork)	byte
11	VolumeNumber	byte
12	DirectoryBase	long (Lo-Hi)
16	LastRecordSeen	word (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	NumberOfLocks	word (Lo-Hi)
...repeats NumberOfLocks times...		
12	LoggedCount	word (Lo-Hi)
14	ShareableLockCount	word (Lo-Hi)
16	RecordStart	long (Lo-Hi)
20	RecordEnd	long (Lo-Hi)
24	LogicalConnectionNumber	word (Lo-Hi)
26	TaskNumber	word (Lo-Hi)
28	LockType	word (Lo-Hi)

## Parameters

### *DataStreamNumber*

(Request) Specifies the data stream number for the Macintosh name space:

- 0 Resource fork (or DOS)
- 1 Data fork

### *VolumeNumber*

(Request) Specifies the volume on which the file exists.

### *DirectoryBase*

(Request) Specifies the directory entry number for the file or directory path (obtained from calling **File Search Initialize** 0x2222 62).

### *LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *NumberOfLocks*

(Reply) Specifies the number of physical record locks.

### *LoggedCount*

(Reply) Specifies the number of tasks that have the record logged.

### *ShareableLockCount*

(Reply) Specifies the number of tasks that have the record locked in a shareable state.

### *RecordStart*

(Reply) Specifies the byte offset where the record begins within the file.

### *RecordEnd*

(Reply) Specifies the byte offset where the record ends within the file.

### *LogicalConnectionNumber*

(Reply) Specifies the logical connection number for the connection that has the record exclusively locked.

### *TaskNumber*

(Reply) Specifies the task number within the logical connection that has the file exclusively locked.

### *LockType*

(Reply) Specifies the bits indicating the file's lock status:

- 0x00 Not locked
- 0xFE Locked by a file lock
- 0xFF Locked by Begin Share File Set

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number
255	0xFF	No Files Found

## Remarks

You must have console operator privileges to call **Get Physical Record Locks By File**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

## See Also

**Get Physical Record Locks By Connection And File 0x2222 23 237 (page 930)**

# Get Physical Record Locks By File (old) 0x2222 23 222

Returns a logical connection's physical record locks within a file.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5 + PathLen)	word (Hi-Lo)
9	SubFunctionCode (222)	byte
10	LastRecordSeen	word
12	DirectoryHandle	byte
13	PathLen	byte
14	FilePath	byte[PathLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word
10	NumberOfLocks	byte
11	Reserved	byte
...repeats NumberOfLocks times...		
12	LoggedCount	word
14	ShareableLockCount	word
16	RecordStart	long
20	RecordEnd	long
24	LogicalConnectionNumber	word
26	TaskNumber	byte
27	LockType	byte

## Parameters

### *LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

### *DirectoryHandle*

(Request) Specifies the file path that is relative to this directory.

### *PathLen*

(Request) Specifies the length of *FilePath*.

### *FilePath*

(Request) Specifies the path and name of the file for which information is being requested.

### *NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

### *NumberOfLocks*

(Reply) Specifies the number of physical record locks.

### *LoggedCount*

(Reply) Specifies the number of tasks that have the record logged.

### *ShareableLockCount*

(Reply) Specifies the number of tasks that have the record locked in a shareable state.

### *RecordStart*

(Reply) Specifies the byte offset where the record begins within the file.

### *RecordEnd*

(Reply) Specifies the byte offset where the record ends within the file.

### *LogicalConnectionNumber*

(Reply) Specifies the logical connection number for the connection that has the record exclusively locked.

### *TaskNumber*

(Reply) Specifies the task number within the logical connection that has the file exclusively locked.

### *LockType*

(Reply) Specifies the bits indicating the file's lock status:

0x00 Not locked

0xFE Locked by a file lock

0xFF Locked by Begin Share File Set



# Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights
253	0xFD	Bad Station Number
255	0xFF	No Files Found

# Remarks

You must have console operator privileges to call **Get Physical Record Locks By File (old)**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

# See Also

**Get Physical Record Locks By Connection And File 0x2222 23 237 (page 930), Get Physical Record Locks By File 0x2222 23 238 (page 936), Get Physical Record Locks By Connection And File (old) 0x2222 23 221 (page 933)**

# Get Semaphore Information 0x2222 23 242

Returns information about a single semaphore.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + SemaphoreNameLen)	word (Hi-Lo)
9	SubFunctionCode (242)	byte
10	LastRecordSeen	word (Lo-Hi)
12	SemaphoreNameLen	byte
13	SemaphoreName	byte[SemaphoreNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Lo-Hi)
10	OpenCount	word (Lo-Hi)
12	SemaphoreValue	word (Lo-Hi)
14	NumberOfRecords	word (Lo-Hi)
... repeats NumberOfRecords times. ...		
16	LogicalConnectionNumber	word (Lo-Hi)
18	TaskNumber	word (Lo-Hi)

## Parameters

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

*SemaphoreNameLen*

(Request) Specifies the length of *SemaphoreName*.

*SemaphoreName*

(Request) Specifies the name that identifies the semaphore.

*NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

*OpenCount*

(Reply) Specifies the number of logical connections that have this semaphore open.

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore (-127...128).

*NumberOfRecords*

(Reply) Specifies the number of semaphore records.

*LogicalConnectionNumber*

(Reply) Specifies the logical connection number for the connection that is using the semaphore.

*TaskNumber*

(Reply) Specifies the task number within the logical connection that has the semaphore open.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

Call **Get Semaphore Information** iteratively when a logical connection has more semaphores than can be placed in the reply message.

You must have console operator privileges to call **Get Semaphore Information**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

Negative *SemaphoreValue* values are usually interpreted as the number of workstations that are waiting for the service represented by the semaphore. Positive *SemaphoreValue* values are usually interpreted as the number of free resources that are available in the resource pool governed by the semaphore.

## See Also

**Get Connection's Semaphores 0x2222 23 241 (page 872)**

# Get Semaphore Information (old) 0x2222 23 226

Returns information about a single semaphore.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (4 + SemaphoreNameLen)	word (Hi-Lo)
9	SubFunctionCode (226)	byte
10	LastRecordSeen	word
12	SemaphoreNameLen	byte
13	SemaphoreName	byte[SemaphoreNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	NextRequestRecord	word (Hi-Lo)
10	OpenCount	word (Hi-Lo)
12	SemaphoreValue	byte
13	NumberOfRecords	byte
...repeats NumberOfRecords times. ...		
14	LogicalConnectionNumber	word (Hi-Lo)
16	TaskNumber	byte

## Parameters

*LastRecordSeen*

(Request) Specifies the last record for which information is returned (set to zero initially).

*SemaphoreNameLen*

(Request) Specifies the length of *SemaphoreName*.

*SemaphoreName*

(Request) Specifies the name that identifies the semaphore.

*NextRequestRecord*

(Reply) Specifies the next value for *LastRecordSeen*.

*OpenCount*

(Reply) Specifies the number of logical connections that have this semaphore open.

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore (-127...128).

*NumberOfRecords*

(Reply) Specifies the number of semaphore records.

*LogicalConnectionNumber*

(Reply) Specifies the logical connection number for the connection that is using the semaphore.

*TaskNumber*

(Reply) Specifies the task number within the logical connection that has the semaphore open.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

Call **Get Semaphore Information (old)** iteratively when a logical connection has more semaphores than can be placed in the reply message.

You must have console operator privileges to call **Get Semaphore Information (old)**.

For subsequent requests, set *LastRecordSeen* to the value returned in the previous request in *NextRequestRecord*. If *NextRequestRecord* is zero, the server has passed all information back to the requesting client.

Negative *SemaphoreValue* values are usually interpreted as the number of workstations that are waiting for the service represented by the semaphore. Positive *SemaphoreValue* values are usually interpreted as the number of free resources that are available in the resource pool governed by the semaphore.

## See Also

**Get Semaphore Information 0x2222 23 242 (page 942)**

# Get Transaction Tracking Statistics 0x2222 23 213

Returns statistical information about a file server's transaction tracking system.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (213)	byte
10	LastRecordSeen	word
12	SemaphoreNameLen	byte
13	SemaphoreName	byte[SemaphoreNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long (Hi-Lo)
12	TransactionTrackingSupported	byte
13	TransactionTrackingEnabled	byte
14	TransactionVolumeNumber	word
16	ConfiguredMaxSimultaneousTransactions	word
18	ActualMaxSimultaneousTransactions	word
20	CurrentTransactionCount	word
22	TotalTransactionsPerformed	long
26	TotalWriteTransactionsPerformed	long
30	TotalTransactionsBackedOut	long
34	TotalUnfilledBackoutRequests	word
36	TransactionDiskSpace	word
38	TransactionFATALlocations	long

Offset	Content	Type
42	TransactionFileSizeChanges	long
46	TransactionFilesTruncated	long
50	NumberOfEntries	byte
...repeats NumberOfEntries times. ...		
51	ConnectionNumber	byte
52	TaskNumber	byte

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *TransactionTrackingSupported*

(Reply) Specifies whether the server supports transaction tracking:

0 File server is not an SFT file server; all other fields are undefined

### *TransactionTrackingEnabled*

(Reply) Specifies whether the server's transaction tracking system is enabled.

### *TransactionVolumeNumber*

(Reply) Specifies the transaction's volume in the volume table maintained by the file server.

### *ConfiguredMaxSimultaneousTransactions*

(Reply) Specifies the maximum number of transactions that the server can simultaneously track.

### *ActualMaxSimultaneousTransactions*

(Reply) Specifies the maximum number of transactions that have simultaneously occurred since the server was brought up.

### *CurrentTransactionCount*

(Reply) Specifies the number of transaction that are currently in progress.

### *TotalTransactionPerformed*

(Reply) Specifies the total transactions that were performed by the server since it was brought up.

### *TotalWriteTransactionsPerformed*

(Reply) Specifies the total number of transactions that required the file server to track file changes.

### *TotalTransactionsBackedOut*

(Reply) Specifies the total number of transactions that TTS has backed out since the file server was brought up.

### *TotalUnfilledBackoutRequests*

(Reply) Specifies the number of transaction backout requests that failed because TTS was disabled.

### *TransactionDiskSpace*

(Reply) Specifies the number of disk blocks being used by TTS (1 block = 4,096 bytes).

### *TransactionFATALlocations*

(Reply) Specifies the number of blocks that have been allocated for FAT sectors of files being tracked since the server was brought up (1 block = 4,096 bytes).

### *TransactionFileSizeChanges*

(Reply) Specifies the number of times since the server was brought up that files being tracked changed their size within a transaction.

### *TransactionFilesTruncated*

(Reply) Specifies the number of times since the server was brought up that files being tracked have been truncated within a transaction.

### *ConnectionNumber*

(Reply) Specifies the logical connection number for the connection involved in a transaction.

### *TaskNumber*

(Reply) Specifies the task number within the logical connection that is involved in a transaction.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

Transaction tracking is disabled when the volume on which the transaction files reside is filled or when you call **Disable Transaction Tracking** 0x2222 23 207.

The volume table contains information about each volume on the file server. A file server running NetWare 2.1 and above can accommodate up to 32 volumes (0...31).

When *TotalTransactionPerformed* reaches 0xFFFFFFFF, it wraps back to zero.



If a workstation requests a transaction but does not actually modify (write) a file, TTS ignores the transaction.

TTS backs out transactions if a workstation requests a backout or if a workstation fails during a transaction. *TotalTransactionBackedOut* includes backout requests that the file server could not perform because TTS was disabled.

# Get Volume Information 0x2222 23 233

Returns information about a specified volume.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (233)	byte
10	VolumeNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long
12	VolumeNumber	byte
13	LogicalDriveNumber	byte
14	BlockSize	word
16	StartingBlock	word
18	TotalBlocks	word
20	FreeBlocks	word
22	TotalDirectoryEntries	word
24	FreeDirectoryEntries	word
26	ActualMaxUsedDirectoryEntries	word
28	VolumeHashedFlag	byte
29	VolumeCachedFlag	byte
30	VolumeRemovableFlag	byte
31	VolumeMountedFlag	byte
32	VolumeName	byte[16]

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *VolumeNumber*

(Reply) Specifies the volume in a volume table on the file server.

### *LogicalDriveNumber*

(Reply) Specifies the logical drive number of the drive on which the volume exists.

### *BlockSize*

(Reply) Specifies the number of 512-byte sectors that are contained in each block of the specified volume.

### *StartingBlock*

(Reply) Specifies the number of the volume's first block.

### *TotalBlocks*

(Reply) Specifies the total number of blocks in the specified volume.

### *FreeBlocks*

(Reply) Specifies the total number of unused blocks in the specified volume.

### *TotalDirectoryEntries*

(Reply) Specifies the number of directory slots that are allocated for the specified volume.

### *FreeDirectoryEntries*

(Reply) Specifies the number of unused directory slots.

### *ActualMaxUsedDirectoryEntries*

(Reply) Specifies the most directory slots that were simultaneously used on the volume.

### *VolumeHashedFlag*

(Reply) Specifies whether the volume is hashed in file server memory:

0 Not hashed

### *VolumeCacheFlag*

(Reply) Specifies whether the volume is cached in file server memory:

0 Volume not cached

### *VolumeRemovableFlag*

(Reply) Specifies whether the volume is physically mounted in the file server:

0 Volume is not mounted

### *VolumeName*

(Reply) Specifies the name given to the volume (from 1 to 16 characters).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
198	0xC6	No Console Rights

## Remarks

You must have console operator privileges to call **Get Volume Information**.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

The volume table contains information about each volume on the file server. A file server running NetWare 2.1 and above can accommodate up to 32 volumes (0...31).

*VolumeName* cannot contain spaces or the characters: \*, ?, :, /, or \. If it less than 16 characters long, the remaining characters must be NULL.

# Read Disk Cache Statistics 0x2222 23 214

Returns statistical information about a file server's caching.

NetWare Servers: 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFunctionCode (214)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long
12	CacheBufferCount	word
14	CacheBufferSize	word
16	DirtyCacheBuffers	word
18	CacheReadRequests	long
22	CacheWriteRequests	long
26	CacheHits	long
30	CacheMisses	long
34	PhysicalReadRequests	long
38	PhysicalWriteRequests	long
42	PhysicalReadErrors	word
44	PhysicalWriteErrors	word
46	CacheGetRequests	long
50	CacheFullWriteRequests	long
54	CachePartialWriteRequests	long
58	BackgroundDirtyWrites	long

Offset	Content	Type
62	BackgroundAgedWrites	long
66	TotalCacheWrites	long
70	CacheAllocations	long
74	ThrashingCount	word
76	LRUBlockWasDirty	word
78	ReadBeyondWrite	word
80	FragmentWriteOccurred	word
82	CacheHitOnUnavailableBlock	word
84	CacheBlockScrapped	word

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *CacheBufferCount*

(Reply) Specifies the number of cache buffers in the server.

### *CacheBufferSize*

(Reply) Specifies the number of bytes in the cache buffer.

### *DirtyCacheBuffers*

(Reply) Specifies the number of cache buffers that are currently in use.

### *CacheReadRequests*

(Reply) Specifies the number of times the cache software received a request to read data from the disk.

### *CacheWriteRequests*

(Reply) Specifies the number of times the cache software received a request to write data to the disk.

### *CacheHits*

(Reply) Specifies the number of times that cache requests were serviced from the existing cache blocks.

### *CacheMisses*

(Reply) Specifies the number of times that cache requests could not be serviced from the existing cache blocks.

### *PhysicalReadReqeusts*

(Reply) Specifies the number of time that the cache software issued a physical read request (as much data as the cache block can hold) to a disk driver.

#### *PhysicalWriteRequests*

(Reply) Specifies the number of times that the cache software issued a physical write request to a disk driver.

#### *PhysicalReadErrors*

(Reply) Specifies the number of times that the cache software received an error from the disk driver on a disk read request.

#### *PhysicalWriteErrors*

(Reply) Specifies the number of times that the cache software received an error from the disk driver on a disk write request.

#### *CacheGetRequests*

(Reply) Specifies the number of times that the cache software received a request to read information from the disk.

#### *CacheFullWriteRequests*

(Reply) Specifies the number of times that the cache software was requested to write information to disk that exactly filled one or more sectors.

#### *CachePartialWriteRequests*

(Reply) Specifies the number of times that the cache software was requested to write information to disk that did not exactly fill a sector (which requires a disk pre-read).

#### *BackgroundDirtyWrites*

(Reply) Specifies the number of times that a cache block that was written to disk was completely filled with information.

#### *BackgroundAgedWrites*

(Reply) Specifies the number of times that the background disk write process wrote a partially filled cache block to disk.

#### *TotalCacheWrites*

(Reply) Specifies the total number of cache buffers written to disk.

#### *CacheAllocations*

(Reply) Specifies the number of times that a cache block was allocated for use.

#### *ThrashingCount*

(Reply) Specifies the number of times that a cache block was not available when a cache block allocation was requested.

#### *LRUBlockWasDirty*

(Reply) Specifies the number of times that the Least-Recently-Used cache block allocation algorithm reclaimed a dirty cache block.

#### *ReadBeyondWrites*

(Reply) Specifies the number of times that a file read request was received for data not yet written to disk (due to file write requests that had not yet filled by cache block, which requires a disk pre-read).

### *FragmentWriteOccurred*

(Reply) Specifies the number of times that a dirty cache block contained noncontiguous sectors of information to be written and the skipped sectors were not preread from the disk.

### *CacheHitOnUnavailableBlock*

(Reply) Specifies the number of times that a cache request could be serviced from an available cache block but the cache buffer could not be used because it was in the process of being written to or read from disk.

### *CacheBlockScrapped*

(Reply) Specifies the number of times that a cache block was scrapped.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

You must have console operator privileges to call **Read Disk Cache Statistics**.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

The volume table contains information about each volume on the file server. A file server running NetWare 2.1 and above can accommodate up to 32 volumes (0...31).

*VolumeName* cannot contain spaces or the characters: \*, ?, :, /, or \. If it less than 16 characters long, the remaining characters must be NULL.

Partially filled cache blocks are written to disk when the block has not been accessed for a significant period of time.

*FragmentWriteOccurred* indicates the number of times multiple disk writes were issued to write out the cache buffer.

A process is put to sleep because it needs a cache block and the LRU cache block had to be written to disk before it could be reused. When the process is awakened after the LRU cache block has been written and freed, the process discovers that while it was asleep, a second process has allocated a different cache block into which the second process has read the information needed by the first sleeping process. In order for the first process to access the information in the cache block allocated by the second process, the first process must free (or scrap) its cache block.



## See Also

[Read Physical Disk Statistics 0x2222 23 216 \(page 958\)](#)

# Read Physical Disk Statistics 0x2222 23 216

Returns statistics about a specified disk.

**NetWare Servers:** 2.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (2)	word (Hi-Lo)
9	SubFunctionCode (216)	byte
10	PhysicalDiskNumber	byte

## Reply Format

Offset	Content	Type
Reply header		
8	SystemIntervalMarker	long
12	PhysicalDiskChannel	byte
13	DriveRemovableFlag	byte
14	PhysicalDriveType	byte
15	ControllerDriveNumber	byte
16	ControllerNumber	byte
17	ControllerType	byte
18	DriveSize	long
22	DriveCylinders	word
24	DriveHeads	byte
25	SectorsPerTrack	byte
26	DriveDefinitionString	byte[64]
90	IOErrorCount	word
92	HotFixTableStart	long
96	HotFixTableSize	word

Offset	Content	Type
98	HotFixBlocksAvailable	word
100	HotFixDisabled	byte

## Parameters

### *SystemIntervalMarker*

(Reply) Specifies the length of time the server has been up.

### *PhysicalDiskChannel*

(Reply) Specifies the disk channel to which the disk unit is attached.

### *DriveRemovableFlag*

(Reply) Specifies whether a disk is removable:

0 Nonremovable

### *PhysicalDriveType*

(Reply) Specifies the type of drive:

1 XT

2 AT

3 SCSI

4 Disk coprocessor

5 PS/2 with MFM Controller

6 PS/2 with ESDI Controller

7 Convergent Technology SBIC

50...255 Value-Added Disk Drive (VADD)

### *ControllerDriveNumber*

(Reply) Specifies the drive number of the disk unit relative to the controller number.

### *ControllerNumber*

(Reply) Specifies the number that identifies the type (make and model) of the disk controller.

### *DriveSize*

(Reply) Specifies the size of the physical drive in blocks (1 block = 4,096 bytes).

### *DriveCylinders*

(Reply) Specifies the number of physical cylinders on the drive.

### *DriveHeads*

(Reply) Specifies the number of disk heads on the drive.

### *SectorsPerTrack*

(Reply) Specifies the number of sectors on each disk track (1 sector = 512 bytes).

### *DriveDefinitionString*

(Reply) Specifies the make and model of the drive (NULL-terminated).

### *IOErrorCount*

(Reply) Specifies the number of times that I/O errors occurred on the disk since the server was brought up.

### *HotFixTableStart*

(Reply) Specifies the first block of the disk Hot Fix Redirection Table (SFT NetWare Level I or above).

### *HotFixTableSize*

(Reply) Specifies the total number of redirection blocks set aside on the disk for Hot Fix redirection.

### *HotFixBlocksAvailable*

(Reply) Specifies the number of redirection blocks that are still available (SFT NetWare Level I or above).

### *HotFixDisabled*

(Reply) Specifies whether Hot Fix is disabled (SFT NetWare Level I or above):

0 Enabled

1 Disabled

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
198	0xC6	No Console Rights

## Remarks

You must have console operator privileges to call **Read Physical disk Statistics**.

*SystemIntervalMarker* is returned in units of approximately 1/18 of a second and is used to determine the amount of time that has elapsed between consecutive requests. (There are 18.2 ticks per second, or one tick every 0.54945054 of a second). When this parameter reaches 0xFFFFFFFF, it wraps back to zero.

*DriveSize* does not include the portion of the disk that is reserved for Hot Fix redirection in the event of media errors.

The redirection table is used to replace bad disk blocks with usable blocks in the event that a media failure occurs on the disk.

Some of all of the blocks specified by *HotFixTableSize* might be in use. This field is meaningful only with SFT NetWare Level I or above.

## See Also

[Read Disk Cache Statistics 0x2222 23 214 \(page 953\)](#)

# Send Console Broadcast 0x2222 23 253

Sends a message to a list of logical connections.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (253)	byte
10	NumberOfStations	byte
11	StationList	long[NumberOfStations]
11 + (NumberOfStations*4)	MessageLen	byte
12 + (NumberOfStations*4)	BroadcastMessage	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

**Send Console Broadcast** can accept messages up to 250 bytes in length.

You must have console operator privileges to call **Send Console Broadcast**.

A message will not reach a station that has disabled broadcasts or is not logged in.

$SubFuncStrucLen = 3 + (NumberOfStations * 4) + MessageLen$

# Send Console Broadcast (old) 0x2222 23 209

Sends a message to a list of logical connections.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen	word (Hi-Lo)
9	SubFunctionCode (209)	byte
10	NumberOfStations	byte
11	StationList	long[NumberOfStations]
11 + (NumberOfStations*4)	MessageLen	byte
12 + (NumberOfStations*4)	BroadcastMessage	byte[MessageLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights
253	0xFD	Bad Station Number

## Remarks

You must have console operator privileges to call **Send Console Broadcast (old)**.

A message will not reach a station that has disabled broadcasts or is not logged in.

*SubFuncStrucLen = 3 + NumberOfStations + MessageLen*

## See Also

**Send Console Broadcast 0x2222 23 253 (page 962)**

# Set File Server Date And Time 0x2222 23 202

Sets the file server's date and time.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (7)	word (Hi-Lo)
9	SubFunctionCode (202)	byte
10	CurrentYear (0 to 99)	byte
11	CurrentMonth (1 to 12)	byte
12	CurrentDay (1 to 31)	byte
13	CurrentHour (0 to 23)	byte
14	CurrentMinute (0 to 59)	byte
15	CurrentSecond (0 to 59)	byte

## Parameters

### *CurrentYear*

(Request) Specifies the year (from 0 to 99, where 80-99 correspond to 1980-1999 and 0-79 correspond to 2000-2079).

### *CurrentMonth*

(Request) Specifies the month (from 1 to 12 corresponding to January through December).

### *CurrentDay*

(Request) Specifies the day (from 1 to 31).

### *CurrentHour*

(Request) Specifies the hour (from 0 to 23 where zero is midnight and 23 is 11 p.m.).

### *CurrentMinute*

(Request) Specifies the minute (from 0 to 59).

### *CurrentSecond*

(Request) Specifies the second (from 0 to 59).



## Return Values

Decimal	Hex	Description
0	0x00	Successful
198	0xC6	No Console Rights

## Remarks

You must have console operator or supervisor privileges to call **Set File Server Date And Time**.

If an invalid value is specified (for example, "250" for *CurrentMonth*), an error is not returned; instead, the file server is set to an undefined but valid date and time.

## See Also

**Get File Server Date And Time 0x2222 20 (page 895)**

# Verify Serialization 0x2222 23 12

Tests the specified server serial number to see if it matches the serial number of the client's server.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (23)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFunctionCode (212)	byte
10	ServerSerialNumber	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Failure

## Remarks

If the serial numbers match, the server returns its server application number for further checking by the client. If the serial numbers do not match, the server destroys the client's service connection.

# 41 Structures

This section describes the Server Environment structures and their fields.

# WaitRecord

Contains information about the current lock state of the target connection.

## Syntax (LockStatus =1)

```
byte    waitingTaskNumber
word    beginAddress (Lo-Hi)
word    endAddress (Lo-Hi)
byte    volumeNumber
word    directoryEntry (Lo-Hi)
byte    lockedName (filename)
```

## Syntax (LockStatus =2)

```
byte    waitingTaskNumber
byte    volumeNumber
word    directoryEntry (Lo-Hi)
byte    lockedName (filename)
```

## Syntax (LockStatus =3)

```
byte    waitingTaskNumber
byte    lockedName (filename)
```

## Syntax (LockStatus =4)

```
byte    waitingTaskNumber
byte    lockedName (filename)
```

# XVIII Statistical

To access Statistical NCPs, use the following:

- ♦ [Chapter 42, “NCPs,” on page 971](#)
- ♦ [Chapter 43, “Structures,” on page 1087](#)



# 42 NCPs

This section describes each of the Statistical NCPs, their Request and Reply formats, and Return Values.

# Active LAN Board List 0x2222 123 20

Returns information about active LAN boards in the server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (20)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	MaxNumOfLans	long (Lo-Hi)
20	ItemsCount	long (Lo-Hi)
24+	BoardNumbers	long[] (Lo-Hi)

## Parameters

*StartNumber*

(Request) Specifies the LAN board to start with.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.



*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*MaxNumOfLans*

(Reply) Specifies the total number of LAN boards.

*ItemsCount*

(Reply) Specifies the number of LAN boards returned to SSGetActiveLANBoardList.

*BoardNumbers*

(Reply) Specifies the first LAN board number, followed by board numbers for each LAN board.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Start Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

To retrieve the rest of the LAN board numbers, call Active LAN Board List again, using the total number of items returned by all previous requests to SSGetActiveLANBoardList plus 1 as *StartNumber*.

# Active Protocol Stacks 0x2222 123 40

Returns protocol stack information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (40)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	MaxNumOfLans	long (Lo-Hi)
20	StackCount	long (Lo-Hi)
24	NextStartNumber	long (Lo-Hi)
24+	AllAttrStruc	structure

## Parameters

### *StartNumber*

(Request) Specifies the number to start with if Active Protocol Stacks is being called iteratively.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *MaxNumOfStacks*

(Reply) Specifies the total number of protocol stacks.

### *StackCount*

(Reply) Specifies the number *StackInfo* structures in the buffer.

### *NextStartNumber*

(Reply) Specifies the start number to use with subsequent requests.

### *StackInfo*

(Reply) Points to the first *StackInfo* structure in the buffer.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Start Number

## Remarks

For the first request, *StartNumber* should be zero. For subsequent requests, use the value in *NextStartNumber*.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# CPU Information 0x2222 123 08

Returns protocol stack information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (08)	byte
10	CPUNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	NumberOfCPUs	long (Lo-Hi)
20+	CPUInformation	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfCPUs*

(Reply) Specifies the number of CPUs in the server.

*CPUInformation*

(Reply) Points to the CPUInformation structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Start Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Enumerate Connection Information from Connection List 0x2222 123 16

Returns connection information about each given connection number.

**NetWare Servers:** 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9 + sizeof(connList))	word (Hi-Lo)
9	SubFuncCode (16)	byte
10	infoMask (variable)	long (Lo-Hi)
14	itemsInList (variable)	long (Lo-Hi)
18+	connList (variable)	long[] (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	itemsReturned	long (Lo-Hi)
20+	connectionInfo	structures

## Parameters

*infoMask*

(Request) Specifies the type of information to be returned.

*itemsInList*

(Request) Specifies the number of connection numbers present in *connList*.

*connList*

(Request) Specifies a list of connection numbers to enumerate for information.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*itemsReturned*

(Reply) Specifies the number of connections for which information was returned.

*connectionInfo*

(Reply) Points to the requested information.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
119	0x77	Insufficient Buffer Size
126	0x7E	Invalid Length

## Remarks

Each connection number will be enumerated for the requested information. Each returned connection item will contain the connection number at a minimum plus additional information as requested in the mask.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

*infoMask* is defined as follows:

Bit	Value	Definition
0	0x00000001	irmTransInfoBit      Transport Information (see <a href="#">netAddr</a> )
1	0x00000002	irmTimeInfoBit      Time Information (see <a href="#">timeInfo</a> )
2	0x00000004	nameInfoBit      Name Information (see <a href="#">nameInfo</a> )

Bit	Value		Definition
3	0x00000008	lockInfoBit	Lock Information (see <a href="#">lockInfo</a> )
4	0x00000010	printInfoBit	Print Information (see <a href="#">printInfo</a> )
5	0x00000020	statsInfoBit	Statistical Information (see <a href="#">statsInfo</a> )
6	0x00000040	acctInfoBit	Accounting Information (see <a href="#">acctngInfo</a> )
7	0x00000080	authInfoBit	Authentication Information (see <a href="#">authInfo</a> )
8	0x00000100	LanguageInfoBit	Language/NCP Encoding Information (see <a href="#">languageInfo</a> )

The LanguageInfoBit (0x100) retrieves information that [Set Connection Language Encoding 0x2222 23 34 \(page 219\)](#) can set.



# Enumerate NCP Service Network Addresses 0x2222 123 17

Enumerates and returns the network address structures that the server is currently accepting NCP requests on.

**NetWare Servers:** 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (167)	byte
10	SearchNumber (variable)	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	serverInfoFlags	word (Lo-Hi)
16	serverGUID	byte[16]
32	nextSearchNumber	long (Lo-Hi)
36	addressesReturned	long (Lo-Hi)
40+	<b>ncpNetworkAddress</b>	structures

## Parameters

*SearchNumber*

(Request) Specifies the starting number and is used to restart multiple replies (if necessary) to retrieve all network address structures (set to zero initially).

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *serverInfoFlags*

(Reply) Specifies whether the server is a cluster member:

0x00000001 CLUSTER\_MEMBER\_BIT

### *serverGUID*

(Reply) Specifies the server's GUID number.

### *nextSearchNumber*

(Reply) Specifies if all the network addresses were returned:

0 All network address structures were returned

nonzero More network address structures exist

### *addressesReturned*

(Reply) Specifies the number of ncpNetworkAddress structures that were returned by the server.

### *ncpNetworkAddress*

(Reply) Points to ncpNetworkAddress.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
119	0x77	Insufficient Buffer Size
126	0x7E	Invalid Length

## Remarks

The server's GUID is automatically generated by the OS the first time the server is brought up. Until a LAN board is loaded for the first time, this number will be all zeros (the Ethernet address of a LAN board is part of the GUID). The server GUID is kept in the registry so the initial GUID created for the server will be preserved across reboots unless the registry is corrupted or deleted. If the registry is corrupted or deleted, the start-up code will automatically generate a new server GUID as if this were the first boot.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Garbage Collection Information 0x2222 123 07

Returns information about garbage collection for a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (07)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	FailedAllocReqCnt	long (Lo-Hi)
20	NumberOfAllocs	long (Lo-Hi)
24	NoMoreMemAvlCnt	long (Lo-Hi)
28	NumOfGarbageColl	long (Lo-Hi)
32	FoundSomeMem	long (Lo-Hi)
36	NumOfChecks	long (Lo-Hi)

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *FailedAllocReqCnt*

(Reply) Specifies the number of times memory allocation failed since the server was brought up.

### *NumberOfAllocs*

(Reply) Specifies the number of memory allocations made since the server was brought up.

### *NoMoreMemAvlCnt*

(Reply) Specifies the number of times that allocation failed since the server was brought up because there was no memory available.

### *NumOfGarbageColl*

(Reply) Specifies the number of time garbage collection was invoked since the server was brought up.

### *FoundSomeMem*

(Reply) Specifies the number of time garbage collection reclaimed memory.

### *NumOfChecks*

(Reply) Specifies the number of times since the server was brought up that garbage collection checked for memory.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Active Connection List by Type 0x2222 123 14

Returns a list of all connections of a given type on the currently connection server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (14)	byte
10	StartConnNumber	long (Lo-Hi)
14	ConnectionType	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	ActiveConnBitList	byte[512]

## Parameters

*StartConnNumber*

(Request) Specifies the starting connection number from which to begin the list.

*ConnectionType*

(Request) Specifies type of connection to include in the list.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *ActiveConnBitList*

(Reply) Specifies the active connections.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
253	0xFD	Invalid Connection
255	0xFF	Failure or Invalid Start Number

## Remarks

*ConnectionType* can have the following values:

- 1 Included in CLIB for backward compatibility
- 2 NCP\_CONNECTION\_TYPE
- 3 NLM\_CONNECTION\_TYPE
- 4 AFP\_CONNECTION\_TYPE
- 5 FTAM\_CONNECTION\_TYPE
- 6 ANCP\_CONNECTION\_TYPE
- 7 ACT\_CONNECTION\_TYPE
- 8 SMB\_CONNECTION\_TYPE
- 9 WINSOCK\_CONNECTION\_TYPE

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

An array of 512 bytes is returned for *ActiveConnBitList*, which sets a bit for each active connection. The connection number is determined by its position in the array.

# Get Cache Information 0x2222 123 01

Returns cache buffer information for the current connection.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (01)	byte
10	VersionNumber (01)	byte
11	RevisionNumber (00)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	Counters	structure
16 + [Counters]	ExtraCacheCntrs	structure
16 + [Counters + ExtraCacheCntrs]	MemoryCounters	structure
16 + [Counters + ExtraCacheCntrs + MemoryCounters]	TrendCounters	structure
16 + [Counters + ExtraCacheCntrs + MemoryCounters + TrendCounters]	CacheInfo	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*Counters*

(Reply) Points to the Counters structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

If *SubFuncStrucLen* is passed as 1, the reply buffer will contain the following structures:

CacheCounters

MemoryCounters

TrendCounters (without the last two fields)

CacheInfo

If *SubFuncStrucLen* is passed as 3, you will need to adjust the size of your reply buffer to contain additional information and the reply buffer will contain the following structures:

CacheCounters

ExtraCacheCounters

MemoryCounters

TrendCounters

CacheInfo

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

An array of 512 bytes is returned for *ActiveConnBitList*, which sets a bit for each active connection. The connection number is determined by its position in the array.



# Get Compression and Decompression Time and Counts 0x2222 123 72

Returns current times and counts on a selected volume.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (72)	byte
10	VolumeNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CmpHiTickHigh	long (Lo-Hi)
12	CmpHiTickCnt	long (Lo-Hi)
16	CmpByteInCount	long (Lo-Hi)
20	CmpByteOutCnt	long (Lo-Hi)
24	CmpHiByteInCnt	long (Lo-Hi)
28	CmpHiByteOutCnt	long (Lo-Hi)
32	DeCpHiTickHigh	long (Lo-Hi)
36	DeCpHiTickCnt	long (Lo-Hi)
40	DeCpByteInCount	long (Lo-Hi)
44	DeCpByteOutCnt	long (Lo-Hi)
48	DeCpHiByteInCnt	long (Lo-Hi)
52	DeCpHiByteOutCnt	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Get Current Compressing File 0x2222 123 70

Returns the file currently being compressed on a selected volume.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (70)	byte
10	VolumeNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	ParentDirEntry	long (Lo-Hi)
12	DirectoryEntry	long (Lo-Hi)
16	compressionStage	long (Lo-Hi)
20	ttlIntermediateBlks	long (Lo-Hi)
24	ttlCompBlks	long (Lo-Hi)
28	curlIntermediateBlks	long (Lo-Hi)
32	curCompBlks	long (Lo-Hi)
36	curCompBlks	long (Lo-Hi)
40	curlInitialBlks	long (Lo-Hi)
44	fileFlags	long (Lo-Hi)
48	projectedCompSize	long (Lo-Hi)
52	originalSize	long (Lo-Hi)
56	compressVolume	long (Lo-Hi)
60	DOSFileNameLen	byte
61	DOSFileName	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

# Get Current DeCompressing File Info List 0x2222 123 71

Returns a list of files and being decompressed on a selected volume.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (71)	byte
10	VolumeNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	FileListCount	long (Lo-Hi)
12	FileInfoStruct	structure[32]

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*FileListCount* contains the number of valid FileInfoStruct entries. Up to 32 files and their information can be passed.

# Get Directory Cache Information 0x2222 123 12

Returns directory cache information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (12)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	<b>DirCacheInfo</b>	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*DirCacheInfo*

(Reply) Points to the DirCacheInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get File Server Information 0x2222 123 02

Returns information about a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (02)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NCPStaInUseCnt	long (Lo-Hi)
20	NCPPeakStaInUse	long (Lo-Hi)
24	NumOfNCPReqs	long (Lo-Hi)
28	ServerUtilization	long (Lo-Hi)
32++	ServerInfo	structure
yy++	FileServerCounters	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.



### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *NCPStaInUse*

(Reply) Specifies the number of workstations connected to the server.

### *NCPPeakStaInUse*

(Reply) Specifies the maximum number of workstations connected at one time since the server was brought up.

### *NumOfNCPReqs*

(Reply) Specifies the number of NCP requests received by the server since it was brought up.

### *ServerUtilization*

(Reply) Specifies the current percentage of CPU utilization for the server.

### *ServerInfo*

(Reply) Points to the ServerInfo structure.

### *FileServerCounters*

(Reply) Points to the FileServerCounters structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

For NetWare 5.1 (Support Pack 6) and NetWare 6 (Support Pack 3) and later versions, the fields in **FileServerCounters** (page 1103) return 0 on an IP-only server because these fields contain IPX-specific information.

# Get General Router and SAP Information 0x2222 123 50

Returns router and SAP information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (50)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	RIPSocketNumber	long (Lo-Hi)
20	RouterDownFlag	long (Lo-Hi)
24	TrackOnFlag	long (Lo-Hi)
28	ExtRouterActiveFlag	long (Lo-Hi)
32	SAPSocketNumber	long (Lo-Hi)
36	RpyNearestSrvFlag	long (Lo-Hi)

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *RIPSocketNumber*

(Reply) Specifies the router socket number.

### *RouterDownFlag*

(Reply) Specifies whether the internal router is up.

### *TrackOnFlag*

(Reply) Specifies whether router tracking is active (the console operator issued the TRACK ON console command).

### *ExtRouterActiveFlag*

(Reply) Specifies whether an external router is active.

### *SAPSocketNumber*

(Reply) Specifies the number of the socket that receives SAP packets.

### *RpyNearestSrvFlag*

(Reply) Specifies whether the server will respond to GetNearestServer.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Known Networks Information 0x2222 123 53

Returns information about known networks.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (53)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NumberOfEntries	long (Lo-Hi)
20	KnownRoutes	structure

## Parameters

*StartNumber*

(Request) Specifies the number of the network to begin returning information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfEntries*

(Reply) Specifies the number of entries for which information is returned.

*KnownRoutes*

(Reply) Points to the KnownRoutes structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Start Number
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Known Servers Information 0x2222 123 56

Returns information about known servers.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (56)	byte
10	StartNumber	long (Lo-Hi)
14	ServerType	long (Hi-Lo)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NumberOfEntries	long (Lo-Hi)
20+	<b>KnownServStruc</b>	structure

## Parameters

*StartNumber*

(Request) Specifies the number of the network to begin returning information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfEntries*

(Reply) Specifies the number of entries for which information is returned.

*KnownServStruc*

(Reply) Points to the KnownServStruc structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Start Number
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Loaded Media Number 0x2222 123 47

Returns information about loaded media.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (47)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	MaxNumOfMedia	long (Lo-Hi)
20	MediaListCount	long (Lo-Hi)
24+	MediaList	long[] (Lo-Hi)

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.



*MaxNumOfMedia*

(Reply) Specifies the maximum number of media.

*MediaListCount*

(Reply) Specifies the count of media loaded in the server.

*MediaList*

(Reply) Specifies the list of media loaded in the server.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Media Manager Object Children's List 0x2222 123 32

Returns a list of children belonging to a given media manager parent object.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (13)	word (Hi-Lo)
9	SubFuncCode (32)	byte
10	StartNumber	long (Lo-Hi)
14	ObjectType	long (Lo-Hi)
18	ParentObjectNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NextStartNum	long (Lo-Hi)
20	ObjectCount	long (Lo-Hi)
24+	Objects	long[] (Lo-Hi)

## Parameters

*StartNumber*

(Request) Specifies the number to start with (set to zero initially).

*ObjectType*

(Request) Specifies the type of object to return information for.

### *ParentObjectNumber*

(Request) Specifies the object number of the media manager object for which you want a list of children.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *nextStartNum*

(Reply) Specifies the number to pass as the *StartNumber* on subsequent requests.

### *ObjectCount*

(Reply) Specifies the number of child objects that are in the buffer.

### *Objects*

(Reply) Specifies the ID number of the first media manager child object with more ID numbers following.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
121	0x79	Invalid Start Number, Object Type, or Parent Object Number
126	0x7E	Invalid Length

## Remarks

For subsequent requests, set *StartNumber* to the value returned in *nextStartNum* of the *GMMChildListStructure*.

*ObjectType* can have the following values:

- 0 ADAPTER\_OBJECT
- 1 CHANGER\_OBJECT
- 2 RDEVICE\_OBJECT
- 3 DEVICE\_OBJECT
- 4 RMEDIA\_OBJECT
- 5 PARTITION\_OBJECT
- 6 SLOT\_OBJECT

- 7 HOTFIX\_OBJECT
- 8 MIRROR\_OBJECT
- 9 PARITY\_OBJECT
- 10 VOLUME\_SEG\_OBJECT
- 11 VOLUME\_OBJECT
- 12 CLONE\_OBJECT
- 13 FMEDIA\_OBJECT
- 14 UNKNOWN\_OBJECT

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

When *nextStartNum* is zero, all the information has been processed.

# Get Media Manager Object Information 0x2222 123 30

Returns information about media manager objects.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (30)	byte
10	ObjectNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16+	ObjectInformation	structure

## Parameters

### *ObjectNumber*

(Request) Specifies the object number of the media manager object that you requested information for.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*ObjectInformation*

(Reply) Points to the **GenericInfoDef** structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Media Manager Objects List 0x2222 123 31

Returns a list of media manager objects.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (31)	byte
10	StartNumber	long (Lo-Hi)
14	ObjectType	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	nextStartNum	long (Lo-Hi)
20	ObjectCount	long (Lo-Hi)
24+	Objects	long[] (Lo-Hi)

## Parameters

*StartNumber*

(Request) Specifies the number to start with (set to -1 initially).

*ObjectType*

(Request) Specifies the type of object to return information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *nextStartNum*

(Reply) Specifies the number to be passed as the *StartNumber* for the next request.

### *ObjectCount*

(Reply) Specifies the number of media manager objects in the buffer.

### *Objects*

(Reply) Specifies the ID number of the first media manager object with more ID numbers following.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
121	0x79	Invalid Start Number or Object Type
126	0x7E	Invalid Length

## Remarks

For subsequent requests, set *StartNumber* to the value returned in *nextStartNum* of *GetMMObjectLists* structure.

*ObjectType* can have the following values:

- 0 ADAPTER\_OBJECT
- 1 CHANGER\_OBJECT
- 2 RDEVICE\_OBJECT
- 3 DEVICE\_OBJECT
- 4 RMECIA\_OBJECT
- 5 PARTITION\_OBJECT
- 6 SLOT\_OBJECT
- 7 HOTFIX\_OBJECT
- 8 MIRROR\_OBJECT
- 9 PARITY\_OBJECT
- 10 VOLUME\_SEG\_OBJECT
- 11 VOLUME\_OBJECT
- 12 CLONE\_OBJECT



- 13 FMEDIA\_OBJECT
- 14 UNKNOWN\_OBJECT

When *nextStartNum* is zero, all the information has been processed.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Media Name by Media Number 0x2222 123 46

Returns a media name for a given media number.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (46)	byte
10	MediaNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	MediaNameLen	byte
17	MediaName	byte[]

## Parameters

*MediaNumber*

(Request) Specifies the ID number of the media for which you want a name.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*MediaNameLen*

(Reply) Specifies the length of the media name.

*MediaName*

(Reply) Specifies the first byte of the media name.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Network Router Information 0x2222 123 51

Returns information about network routing on a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (51)	byte
10	NetworkNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	NetIDNumber	long (Hi-Lo)
20	HopsToNetCount	word (Hi-Lo)
22	NetStatus	word (Hi-Lo)
24	TimeToNet	word (Lo-Hi)

## Parameters

*NetworkNumber*

(Request) Specifies the network to return information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NetIDNumber*

(Reply) Specifies the network ID number.

*HopsToNetCount*

(Reply) Specifies the number of hops to the network.

*HopsToNetCount*

(Reply) Specifies the number of hops to the network.

*NetStatus*

(Reply) Specifies the status of the network.

*TimeToNet*

(Reply) Specifies the number of ticks to the network (round trip).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Network Number
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Network Routers Information 0x2222 123 52

Returns information about the routers on a network.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (51)	byte
10	NetworkNumber	long (Hi-Lo)
14	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NumberOfEntries	long (Hi-Lo)
20+	<b>RoutersInfo</b>	structure

## Parameters

*NetworkNumber*

(Request) Specifies the network to return information for.

*StartNumber*

(Request) Specifies the number to start with.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfEntries*

(Reply) Specifies the number of structures in the buffer.

*RoutersInfo*

(Reply) Points to the first RoutersInfo structure with more structures following.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Network Number
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get NLM Loaded List 0x2222 123 10

Returns a list of NLMs that are running on the server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (10)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	unused	word (Lo-Hi)
16	NLMcount	long (Lo-Hi)
20	NLMsInList	long (Lo-Hi)
24+	NLMNumbers	long[] (Lo-Hi)

## Parameters

*StartNumber*

(Request) Specifies the NLM number to start with.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.



#### *VConsoleRevision*

(Reply) Specifies the console version revision number.

#### *unused*

(Reply) Is reserved for future use.

#### *NLMcount*

(Reply) Specifies the total number of NLMs that are loaded on the server.

#### *NLMsInList*

(Reply) Specifies the number of NLMs that are returned in the NLM list (maximum of 130 NLMs).

#### *NLMNumbers*

(Reply) Specifies a list containing the numbers assigned to NLMs that are loaded on the server.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

If *NLMcount* is greater than *NLMsInList*, there are more NLMs that could not be listed in the buffer. To obtain the rest of the NLMs, call **Get NLM Loaded List** again using the last NLM number in the last list as the *StartNumber*.

# Get NLM Resource Tag List 0x2222 123 15

Returns information about the resources used by NLMs on a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (10)	byte
10	NLMNumber	long (Lo-Hi)
14	NLMStartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	TtlNumOfRTags	long (Lo-Hi)
20	CurNumOfRTags	long (Lo-Hi)
24+	<b>RTagStructure</b>	structure

## Parameters

*NLMNumber*

(Request) Specifies the assigned to the NLM by the OS when the NLM was loaded.

*NLMStartNumber*

(Request) Specifies the resource tag to start with.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*TtlNumOfRTags*

(Reply) Specifies the total number of resource tags that the NLM is using.

*CurNumOfRTags*

(Reply) Specifies the number of resource tags in the buffer.

*RTagStructure*

(Reply) Points to the first RTagStructure in the buffer with more structures following.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

If there are additional tags that are used by NLMs that could not be listed in the buffer, call **Get NLM Resource Tag List** again using the last NLM number in the last list as the *NLMStartNumber*.

# Get Operating System Version Information 0x2222 123 13

Returns information about the OS on the server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (13)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	OSMajorVersion	byte
17	OSMinorVersion	byte
18	OSRevision	byte
19	AccountVersion	byte
20	VAPVersion	byte
21	QueueingVersion	byte
22	SecurityRestLvl	byte
23	BridgingSupport	byte
24	MaxNumOfVol	long (Lo-Hi)
28	MaxNumOfConn	long (Lo-Hi)
32	MaxNumOfUsers	long (Lo-Hi)
36	MaxNumOfNmeSps	long (Lo-Hi)

Offset	Content	Type
40	MaxNumOfLANS	long (Lo-Hi)
44	MaxNumOfMedias	long (Lo-Hi)
48	MaxNumOfStacks	long (Lo-Hi)
52	MaxDirDepth	long (Lo-Hi)
56	MaxDataStreams	long (Lo-Hi)
60	MaxNumOfSpoolPr	long (Lo-Hi)
64	ServerSerial#	long (Lo-Hi)
68	ServerApp#	word (Lo-Hi)

## Parameters

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *OSMajorVersion*

(Reply) Specifies the major version number of the OS.

### *OSMinorVersion*

(Reply) Specifies the minor version number of the OS.

### *OSRevision*

(Reply) Specifies the version revision letter of the OS.

### *AccountVersion*

(Reply) Specifies the version of the accounting subsystem.

### *VAPVersion*

(Reply) Is not currently used.

### *QueueingVersion*

(Reply) Specifies the queueing version number.

### *SecurityRestLvl*

(Reply) Specifies the security restriction version number.

### *BridgingSupport*

(Reply) Specifies the internet bridge support version number.

### *MaxNumOfVol*

(Reply) Specifies the maximum number of volumes that can be simultaneously mounted on the server.

### *MaxNumOfConn*

(Reply) Specifies the maximum number of connections that can be used simultaneously on the server.

### *MaxNumOfUsers*

(Reply) Specifies the maximum number of simultaneous users allowed on the server.

### *MaxNumOfNmeSps*

(Reply) Specifies the maximum number of name spaces that can be simultaneously loaded on the server.

### *MaxNumOfLANS*

(Reply) Specifies the maximum number of LAN boards that can be used on the server.

### *MaxNumOfMedias*

(Reply) Specifies the maximum number of different media types allowed on the server.

### *MaxNumOfStacks*

(Reply) Specifies the maximum number of protocol stacks that can be used on the server.

### *MaxDirDepth*

(Reply) Specifies the maximum depth of directories that can be used on the server.

### *MaxDataStreams*

(Reply) Specifies the maximum number of data streams that can be used on the server.

### *MaxNumOfSpoolPr*

(Reply) Specifies the maximum number of spool printers that can be used on the server.

### *ServerSerial#*

(Reply) Specifies the serial number of the server.

### *ServerApp#*

(Reply) Is included for backward compatibility.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Protocol Stack Configuration Information 0x2222 123 41

Returns configuration information about the protocols on the server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (41)	byte
10	StackNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	ConfigMajorVN	byte
17	ConfigMinorVN	byte
18	StackMajorVN	byte
19	StackMinorVN	byte
20	ShortStkName	byte[16]
36	StackFullNameLen	byte
37	StackFullNameStr	byte

## Parameters

*StackNumber*

(Request) Specifies the stack number for which configuration information is returned.



*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*ConfigMajorVN*

(Reply) Specifies the major version number of the configuration table.

*ConfigMinorVN*

(Reply) Specifies the minor version number of the configuration table.

*StackMajorVN*

(Reply) Specifies the major version number of the protocol stack.

*StackMinorVN*

(Reply) Specifies the minor version number of the protocol stack.

*ShortStkName*

(Reply) Specifies the short protocol name that is used to register the stack with the LSL.

*StackFullNameLen*

(Reply) Specifies the length of the full protocol name.

*StackFullNameStr*

(Reply) Specifies the first byte of the full name.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

*ShortStkName* can have the following values:

---

VIRTUAL_LAN	Used where no Frame ID/MAC envelope is necessary
LOCALTALK	Apple LocalTalk frame
ETHERNET_II	Ethernet using a DEC Ethernet II envelope
ETHERNET_802.2	Ethernet using an 802.2 envelope
TOKEN-RING	Token-ring (802.5) using an 802.2 envelope
ETHERNET_802.3	IPX 802.3 raw encapsulation
802.4	Token-passing bus envelope
NOVELL_PCN2	Novell's IBM PC Network II envelope
GNET	Gateway's GNET frame envelope
PRONET-10	Proteon's proNET I/O frame envelope
ETHERNET_SNAP	Ethernet (802.3) using an 802.2 envelope with SNAP
TOKEN-RING_SNAP	Token-ring (802.5) using an 802.2 envelope with SNAP
LANPAC_II	Racore's frame envelope
ISDN	Integrated Services Digital Network
NOVELL_RX-NET	Novell's ARCNET envelope
IBM_PCN2_802.2	IBM PCN2 using 802.2 envelope
IBM_PCN2_SNAP	IBM PCN2 using 802.2 with SNAP envelope
OMNINET/4	Corvus's frame envelope
3270_COAXA	Harris Adacom's frame envelope
IP	IP Tunnel frame envelope
FDDI_802.2	FDDI (802.7) using an 802.2 envelope
IVDLAN_802.9	Commtext, Inc.'s frame envelope
DATAOCO_OSI	Dataco's frame envelope
FDDI_SNAP	FDDI (802.7) using 802.2 with a SNAP envelope
IBM_SDLC	SDLC tunnel envelope
PCO_FDDITP	PC Office frame envelope
WAIDNET	Hypercommunications
SLIP	Novell frame envelope
PPP	Novell frame envelope

---

# Get Protocol Stack Custom Information 0x2222 123 43

Returns custom information about a protocol stack on a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (41)	byte
10	StackNumber	long (Lo-Hi)
18	CustomStartNum	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	CustomCount	long (Lo-Hi)
20+	CustomStruc	(long, byte[]) []

## Parameters

*StackNumber*

(Request) Specifies the stack number for which configuration information is returned.

*CustomStartNum*

(Request) Specifies the custom information structure to start with (set to zero initially).

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*CustomCount*

(Reply) Specifies the number of structures in the buffer.

*CustomStruc*

(Reply) Points to the first CustomStruc information, which contains the following information (more structures follow):

*Value* specifying the value of the custom counter.

*Length* specifying the length of custom data.

*CustomData* specifying the first byte of a string describing the custom counter (length of length).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

For subsequent requests, set *CustomStartNum* to the next custom information structure that has not been retrieved. All information has been retrieved when no more information is returned.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Protocol Stack Numbers by LAN Board Number 0x2222 123 45

Returns a list of protocol stack ID numbers for a given LAN board.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (45)	byte
10	LanBdNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	StackNumberCount	long (Lo-Hi)
20+	StackNumbers	long[] (Lo-Hi)

## Parameters

*LanBdNumber*

(Request) Specifies the ID number of the LAN board for which you want a list of protocols.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*StackNumberCount*

(Reply) Specifies the number of protocol stack ID numbers in the buffer.

*StackNumbers*

(Reply) Specifies the first protocol stack ID number in the buffer.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Protocol Stack Numbers by Media Number 0x2222 123 44

Returns a list of protocol stack ID numbers for a given media.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (44)	byte
10	MediaNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	StackNumberCount	long (Lo-Hi)
20+	StackNumbers	long[] (Lo-Hi)

## Parameters

*MediaNumber*

(Request) Specifies the media number (frame number) for which you are requesting protocol stack information.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*StackNumberCount*

(Reply) Specifies the number of protocol stack ID numbers in the buffer.

*StackNumbers*

(Reply) Specifies the first protocol stack ID number in the buffer.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

*MediaNumber* can have the following values:

0	Used where no Frame ID/MAC envelope is necessary
1	Apple LocalTalk frame
2	Ethernet using a DEC Ethernet II envelope
3	Ethernet using an 802.2 envelope
4	Token-ring (802.5) using an 802.2 envelope
5	IPX 802.3 raw encapsulation
6	Token-passing bus envelope
7	Novell's IBM PC Network II envelope
8	Gateway's GNET frame envelope
9	Proteon's proNET I/O frame envelope
10	Ethernet (802.3) using an 802.2 envelope with SNAP
11	Token-ring (802.5) using an 802.2 envelope with SNAP
12	Racore's frame envelope
13	Integrated Services Digital Network
14	Novell's ARCNET envelope



---

15	IBM PCN2 using 802.2 envelope
16	IBM PCN2 using 802.2 with SNAP envelope
17	Corvus's frame envelope
18	Harris Adacom's frame envelope
19	IP Tunnel frame envelope
20	FDDI (802.7) using an 802.2 envelope
21	Commtext, Inc.'s frame envelope
22	Dataco's frame envelope
23	FDDI (802.7) using 802.2 with a SNAP envelope
24	SDLC tunnel envelope
25	PC Office frame envelope
26	Hypercommunications
27	Novell frame envelope

---

# Get Protocol Stack Statistics Information 0x2222 123 42

Returns protocol statistics for a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (42)	byte
10	StackNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	StatMajorVN	word (Lo-Hi)
17	StatMinorVN	byte
18	ComCnts (3)	word (Lo-Hi)
20	CounterMask	long (Lo-Hi)
24	TotalTxPkts	long (Lo-Hi)
28	TotalRxPkts	long (Lo-Hi)
32	IgnoredRxPkts	long (Lo-Hi)
36	CustomCnts	word (Lo-Hi)

## Parameters

*StackNumber*

(Request) Specifies the stack number of the protocol stack for which you want information.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *StatMajorVN*

(Reply) Specifies the major version number of the statistics table.

### *StatMinorVN*

(Reply) Specifies the minor version number of the statistics table.

### *ComCnts*

(Reply) Specifies the number of counters in the fixed portion of the table.

### *CounterMask*

(Reply) Specifies which counters are valid (starting with the right most bit representing the first counter):

- 0 Counter is valid
- 1 Counter is not valid

### *TotalTxPkts*

(Reply) Specifies the total number of packets that were requested to be transmitted.

### *TotalRxPkts*

(Reply) Specifies the total number of packets that were received.

### *IgnoredRxPkts*

(Reply) Specifies the number of incoming packets that were ignored by the stack.

### *CustomCnts*

(Reply) Specifies the number of custom counters for the protocol stack.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Server Information 0x2222 123 54

Returns information about a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (6 + ServerNameLen)	word (Hi-Lo)
9	SubFuncCode (54)	byte
10	ServerType	long (Lo-Hi)
11	ServerNameLen	byte
12	ServerName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	ServerAddress	byte[12]
28	HopsToServer	word (Hi-Lo)

## Parameters

*ServerType*

(Request) Specifies the type of the server.

*ServerNameLen*

(Request) Specifies the length of the server name.

*ServerName*

(Request) Specifies the name of the server.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*ServerAddress*

(Reply) Specifies the node address of the server.

*HopsToServer*

(Reply) Specifies the number of hops to the server.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Server Name or Server Type
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Server Set Categories 0x2222 123 61

Returns information about server set categories.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (61)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	NumberOfSetCategories	long (Lo-Hi)
20	NextStartNumber	long (Lo-Hi)
24++	CategoryName	byte[]

## Parameters

*StartNumber*

(Request) Specifies the number of the server to begin returning information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfSetCategories*

(Reply) Specifies the number of set categories that are registered on the server.

*NextStartNumber*

(Reply) Specifies the next server number to begin returning information for.

*CategoryName*

(Reply) Specifies the category name in ASCIIZ format.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
245	0xF5	Invalid Start Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.



# Get Server Set Commands Information 0x2222 123 60

Returns information about server set commands.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (60)	byte
10	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	TtlNumOfSetCmds	long (Lo-Hi)
20	NextStartNumber	long (Lo-Hi)
24	SetCmdType	long (Lo-Hi)
28	SetCmdCategory	long (Lo-Hi)
32	SetCmdFlags	long (Lo-Hi)
36+	SetCmdName	byte[]
36+[SetCmdName ]	SetCmdValue	byte[]

## Parameters

*StartNumber*

(Request) Specifies the number of the server to begin returning information for.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *TtlNumOfSetCmds*

(Reply) Specifies the total number of set commands.

### *NextStartNumber*

(Reply) Specifies the next server number to begin returning information for.

### *SetCmdType*

(Reply) Specifies the type of the set command.

### *SetCmdCategory*

(Reply) Specifies the set command category.

### *SetCmdFlags*

(Reply) Specifies the set command flags.

### *SetCmdName*

(Reply) Specifies the set command name.

### *SetCmdValue*

(Reply) Specifies the set command value (as a byte string or a long, depending on the set command type).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Start Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Server Set Commands Information By Name 0x2222 123 62

Returns information about server set commands for the specified set parameter name.

**NetWare Servers:** 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1+strlen(SetParmName)+1)	word (Hi-Lo)
9	SubFuncCode (62)	byte
10	SetParmName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	TtlNumOfSetCmds	long (Lo-Hi)
20	NextStartNumber	long (Lo-Hi)
24	SetCmdType	long (Lo-Hi)
28	SetCmdCategory	long (Lo-Hi)
32	SetCmdFlags	long (Lo-Hi)
36+	SetCmdName	byte[]
36+[SetCmdName]	SetCmdValue	byte[]

## Parameters

*SetParmName*

(Request) Specifies a NULL-terminated (and not length preceded) ASCIIZ string that represents the name of the set parameter for which you are requesting information.

#### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

#### *VConsoleVersion*

(Reply) Specifies the console version number.

#### *VConsoleRevision*

(Reply) Specifies the console version revision number.

#### *reserved*

(Reply) Is reserved for future use.

#### *TtlNumOfSetCmds*

(Reply) Specifies the total number of set commands.

#### *NextStartNumber*

(Reply) Specifies the next server number to begin returning information for.

#### *SetCmdType*

(Reply) Specifies the type of the set command.

#### *SetCmdCategory*

(Reply) Specifies the set command category.

#### *SetCmdFlags*

(Reply) Specifies the set command flags.

#### *SetCmdName*

(Reply) Specifies the set command name.

#### *SetCmdValue*

(Reply) Specifies the set command value (as a byte string or a long, depending on the set command type).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Start Number

## Remarks

*SubFuncStrucLen* should be set to 1 plus the length of the *SetParmName* plus an additional 1. The 2 one-byte additions account for the subfunction code and the NULL-terminating byte of *SetParmName*.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Server Sources Information 0x2222 123 55

Returns address information about servers known to the named server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (11)	word (Hi-Lo)
9	SubFuncCode (55)	byte
10	StartNumber	long (Lo-Hi)
14	ServerType	long (Hi-Lo)
18	ServerNameLen	byte
19	ServerName	byte[]

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	NumberOfEntries	long (Lo-Hi)
20+	<b>ServersSrcInfo</b>	structure

## Parameters

*StartNumber*

(Request) Specifies the number of the server to begin returning information for.

*ServerType*

(Request) Specifies the type of server to obtain information for.

*ServerNameLen*

(Request) Specifies the length of the server name.

*ServerName*

(Request) Specifies the name of the server.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumberOfEntries*

(Reply) Specifies the number of structures in the buffer.

*ServersSrcInfo*

(Reply) Points to the first ServersSrcInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
1	0x01	Invalid Server Name or Type
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Volume Information by Level 0x2222 123 34

Returns volume information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (34)	byte
10	VolumeNumber	long (Lo-Hi)
14	InfoLevelNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	InfoLevel	long (Lo-Hi)
20+	volumeInformation	byte[]

## Parameters

*VolumeNumber*

(Request) Specifies the volume number to start with.

*InfoLevelNumber*

(Request) Specifies structure to return:

- 1 **VolInfoDef**
- 2 **VolInfo2Def**

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.



*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*InfoLevel*

(Reply) Specifies the structure that was returned (see *InfoLevelNumber*).

*volumeInformation*

(Reply) Points to the returned volume information structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
152	0x98	Disk Map Error or Invalid Volume
255	0xFF	Failure

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Get Volume Segment List 0x2222 123 33

Returns a list of volume segments for a given volume.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (33)	byte
10	VolumeNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16	NumOfSegments	long (Lo-Hi)
20	Segments	structure

## Parameters

*VolumeNumber*

(Request) Specifies the volume number to start with.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*NumOfSegments*

(Reply) Specifies the number of volume segments on the volume.

*Segments*

(Reply) Points to the first returned volume segment structure (more structures follow, one for each volume segment).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
152	0x98	Disk Map Error or Invalid Volume

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# IPX SPX Information 0x2222 123 06

Returns information about IPX/SPX use on a server.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (06)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved (0)	word (Lo-Hi)
16++	IPXInformation	structure
zz++	SPXInformation	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

### *IpxInformation*

(Reply) Points to the IPXInformation structure.

### *SpxInformation*

(Reply) Points to the SPXInformation structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# LAN Common Counters Information 0x2222 123 22

Returns common statistics for a LAN board.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (22)	byte
10	BoardNumber	long (Lo-Hi)
14	BlockNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	StatMajorVersion	byte
15	StatMinorVersion	byte
16	TotalCommonCnts	long (Lo-Hi)
20	TotalCntBlocks	long (Lo-Hi)
24	CustomCounters	long (Lo-Hi)
28	NextCntBlock	long (Lo-Hi)
32+	CommonLanStruc	structure

## Parameters

*BoardNumber*

(Request) Specifies the board number of the LAN board for which you are requesting information.

### *BlockNumber*

(Request) Specifies the block number to start with (set to zero initially).

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *StatMajorVersion*

(Reply) Specifies the major version number of the generic portion of the statistics table (defined by Novell).

### *StatMinorVersion*

(Reply) Specifies the major version number of the generic portion of the statistics table (defined by Novell).

### *TotalCommonCnts*

(Reply) Specifies the number of LAN common counters.

### *TotalCntBlocks*

(Reply) Specifies the total number of blocks used by LAN common counters for the specified LAN board.

### *CustomCounters*

(Reply) Specifies the number of LAN custom counters.

### *NextCntBlock*

(Reply) Specifies the value to be passed to *BlockNumber* for the next request.

### *CommonLanStruc*

(Reply) Points to the CommonLanStruc structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Board or Block Number

## Remarks

For subsequent requests, set *BlockNumber* to the value returned in *NextCntBlock* of the *GetLANCommonCountersStructure*. When *NextCntBlock* is zero, all information has been returned.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.



# LAN Configuration Information 0x2222 123 21

Returns LAN configuration information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (21)	byte
10	BoardNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	LANConfigInfo	structure

## Parameters

*BoardNumber*

(Request) Specifies the board number of the LAN board for which you are requesting information.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*LANConfigInfo*

(Reply) Points to the LANConfigInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Board or Block Number

## Remarks

For subsequent requests, set *BlockNumber* to the value returned in *NextCntBlock* of the GetLANCommonCountersStructure. When *NextCntBlock* is zero, all information has been returned.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# LAN Custom Counters Information 0x2222 123 23

Returns custom statistics for a LAN board.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (9)	word (Hi-Lo)
9	SubFuncCode (23)	byte
10	BoardNumber	long (Lo-Hi)
14	StartNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	Unused	word
16	NumOfCCinPkt	long (Lo-Hi)
20+	CustomCntsInfo	(long, STRING) []

## Parameters

*BoardNumber*

(Request) Specifies the board number of the LAN board for which you are requesting information.

*StartNumber*

(Request) Specifies the custom counter number to start with (set to zero initially).

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*Unused*

(Reply) Is reserved for future use.

*NumOfCCinPkt*

(Reply) Specifies the number of custom counters used by the LAN driver.

*CustomCntsInfo*

(Reply) Points to the CustomCntsInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Board or Block Number

## Remarks

LAN custom counters keep track of such statistics as the number of raw sends and fatal retransmissions. Each counter has a string describing the counter and a value associated with the counter.

To ensure that all information is returned, call **LAN Custom Counters Information** once. Use the *NumOfCCinPkt* value as the *StartNumber* for the second request. Issue the request a third time to ensure all the information was returned. You can also check by adding the number of custom counters that are returned for each request in *NumOfCCinPkt*. If your last request is greater than or equal to the total number of custom counters, -1 will be returned, which indicates that you've received all the information.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# LAN Name Information 0x2222 123 24

Returns LAN name information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (24)	byte
10	BoardNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	BoardName	structure

## Parameters

### *BoardNumber*

(Request) Specifies the board number of the LAN board for which you are requesting information.

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*BoardName*

(Reply) Points to the BoardName structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Board or Block Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

## See Also

**LAN Configuration Information 0x2222 123 21 (page 1061)**

# LSL Information 0x2222 123 25

Returns LSL information.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (25)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	LSLInformation	structure

## Parameters

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*LSLInformation*

(Reply) Points to the LSLInformation structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.



# LSL Logical Board Statistics 0x2222 123 26

Returns information about the LSL.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (26)	byte
10	LanBoardNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	LogTtlTxPkts	long (Lo-Hi)
20	LogTtlRxPkts	long (Lo-Hi)
24	UnclaimedPkts	long (Lo-Hi)
28	reserved	long (Lo-Hi)

## Parameters

*LanBoardNumber*

(Request) Specifies the board number to return information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *LogTtlTxPkts*

(Reply) Specifies the total number of transmitted packets.

### *LogTtlRxPkts*

(Reply) Specifies the total number of received packets.

### *UnclaimedPkts*

(Reply) Specifies the total number of unclaimed packets.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid LAN Board Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# MLID Board Information 0x2222 123 27

Returns information about the MLID board.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (27)	byte
10	MLIDBoardNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	byte
15	reserved (NetWare 4.x )	byte
	numberOfProtocols (NetWare 5.x only)	
16	MLIDBoardInfo	structure

## Parameters

*MLIDBoardNumber*

(Request) Specifies the board number to return information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*numberOfProtocols*

(Reply) Specifies the number of protocols that are bound to the specified board (NetWare 5.x only).

*MLIDBoardInfo*

(Reply) Points to the MLIDBoardInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid LAN Board Number

## Remarks

For NetWare 4.x, **MLID Board Information** returns an error if a protocol is not bound to the specified board number. Otherwise, the first protocol that is bound to the board is returned. For NetWare 5.x, all protocols that are bound to the board are returned.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# NetWare File Systems Information 0x2222 123 03

Returns information about the NetWare 3.x file system.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (03)	byte
10	FileSystemID	

## Reply Format (NetWare 3.x File System)

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	FileSystemInfo	structure

## Parameters

*FileSystemID*

(Request) Specifies the ID number of the file system for which information is requested (already 1 for 386).

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

*VConsoleRevision*

(Reply) Specifies the console version revision number.

*reserved*

(Reply) Is reserved for future use.

*FileSystemInfo*

(Reply) Points to the FileSystemInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid LAN Board Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# NLM Information 0x2222 123 11

Returns information about NLMs.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (11)	byte
10	NLMNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	NLMInformation	structure
xx+	FileNameLen	byte
yy+	FileName	byte[]
uu+	NameLen	byte
vv+	Name	byte[]
ww+	CopyrightLen	byte
zz+	Copyright	byte[]

## Parameters

*NLMNumber*

(Request) Specifies the number assigned to the NLM by the OS when the NLM was loaded (returned by **Get NLM Loaded List 0x2222 123 10**).

### *CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

### *VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *NLMInformation*

(Reply) Points to the NLMInformation structure.

### *FileNameLen*

(Reply) Specifies the length of the NLM file.

### *FileName*

(Reply) Specifies the name of the NLM file.

### *NameLen*

(Reply) Specifies the length of the NLM.

### *Name*

(Reply) Specifies the name of the NLM.

### *CopyrightLen*

(Reply) Specifies the length of the NLM's copyright.

### *Copyright*

(Reply) Specifies the NLM's copyright name.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid LAN Board Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.



# Packet Burst Information 0x2222 123 05

Returns packet burst information.

NetWare Servers: 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (05)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	PacketBurstInformation	long (Lo-Hi)

## Parameters

- CurrentServerTime*  
(Reply) Specifies the time elapsed since the server was brought up.
- VConsoleVersion*  
(Reply) Specifies the console version number.
- VConsoleRevision*  
(Reply) Specifies the console version revision number.
- reserved*  
(Reply) Is reserved for future use.
- PacketBurstInfo*  
(Reply) Points to the PacketBurstInfo structure.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# User Information 0x2222 123 04

Returns user information for the specified connection.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (04)	byte
10	ConnectionNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16+	UserInformation	structure
yy+	UserNameLen	byte
zz++	UserNameString	byte[]

## Parameters

*ConnectionNumber*

(Request) Specifies the connection number that you are requesting user information for.

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *UserInfo*

(Reply) Points to the UserInfo structure.

### *UserNameLen*

(Reply) Specifies the length of the user name.

### *UserNameString*

(Reply) Specifies the name of the user.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Connection Number

## Remarks

**User Information** succeeds only for fully licensed connections.

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

# Volume Switch Information 0x2222 123 09

Returns information about the number of times various code paths have been taken in the NetWare OS.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (123)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (09)	byte
10	StartItemNumber	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	CurrentServerTime	long (Lo-Hi)
12	VConsoleVersion	byte
13	VConsoleRevision	byte
14	reserved	word (Lo-Hi)
16	TotalLFSCounters	long (Lo-Hi)
20	CurrentLFSCounters	long (Lo-Hi)
24+	LFSCounters	long[] (Lo-Hi)

## Parameters

*StartItemNumber*

(Request) Specifies the number of the counter to start with (included for future use).

*CurrentServerTime*

(Reply) Specifies the time elapsed since the server was brought up.

*VConsoleVersion*

(Reply) Specifies the console version number.

### *VConsoleRevision*

(Reply) Specifies the console version revision number.

### *reserved*

(Reply) Is reserved for future use.

### *TotalLFSCounters*

(Reply) Specifies the total number of LFS counters.

### *CurrentLFSCounters*

(Reply) Specifies the number of LFS counters returned by this request.

### *LFSCounters*

(Reply) Specifies the first LFS counter (followed by others).

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	Invalid Length
255	0xFF	Failure or Invalid Connection Number

## Remarks

*CurrentServerTime* is returned in ticks (approximately 1/18 of a second). When the value reaches 0xFFFFFFFF, it wraps back to zero.

*VConsoleVersion* and *VConsoleRevision* track packet formats.

*LFSCounters* can have the following values:

- 0 CreateDirectory
- 1 DeleteDirectory
- 2 MapPathToDirectoryNumber
- 3 EraseFile
- 4 ModifyDirectoryEntry
- 5 RenameEntry
- 6 GetAccessRights
- 7 GetAccessRightsFromIDs
- 8 MapDirectoryNumberToPath
- 9 GetEntryFromPathStringBa
- 10 GetOtherNameSpaceEntry
- 11 DirectorySearch
- 12 GetExtendedDirectoryInfo
- 13 GetParentDirectoryNumber
- 14 AddTrusteeRights

15 ScanTrusteeRights  
16 DeleteTrusteeRights  
17 PurgeTrustee  
18 FindNextTrusteeReference  
19 ScanUserRestrictionNodes  
20 AddUserRestriction  
21 DeleteUserRestriction  
22 ReturnDirectorySpaceRest  
23 GetActualAvailableDiskSp  
24 CountOwnedFilesAndDirect  
25 ScanDeletedFiles  
26 SalvageDeletedFile  
27 PurgeDeletedFile  
28 OpenFile  
29 CreateFile  
30 CreateAndOpenFile  
31 MigrateFile  
32 DeMigrateFile  
33 MigratedFileInformation  
34 VolumeMigrationInformation  
35 ReadMigratedFileData  
36 ReadFile  
37 WriteFile  
38 ASyncStartReadFile  
39 ASyncDoReadFile  
40 ASyncStartWriteFile  
41 ASyncDoWriteFile  
42 ASyncCheckWriteThrough  
43 NewGetVolumeInfo  
44 MapPathToDirectoryNumberOrPhantom  
45 StationHasAccessRightsGrantedBelow  
46 GetDataStreamLengthsFromPathStringBase  
47 CheckAndGetDirectoryEntry  
48 GetDeletedEntry  
49 GetOriginalNameSpace  
50 GetActualFileSize  
51 VerifyNameSpaceNumber  
52 VerifyDataStreamNumber  
53 CheckVolumeNumber  
54 GetFileSize  
55 ReadFileNoCheck  
56 SetFileTimeAndDateStamp  
57 GetFileHoles  
58 GetHandleInfoData  
59 CloseFile  
60 CommitFile

- 61 GetDirectoryEntry
- 62 CreateDataMigratedFileEntry
- 63 RenameNameSpaceEntry
- 64 CancelFileLockWait
- 65 CheckAndSetSingleFileLock
- 66 ReleaseSingleFileLock
- 67 EnumerateFileLocks
- 68 CheckAndSetFileLocks
- 69 BackoutFileLocks
- 70 UnenumerateFileLocks
- 71 ReleaseFile
- 72 CheckAndSetSingleRecordLock
- 73 ReleaseSingleRecordLock
- 74 EnumerateRecordLocks
- 75 CheckAndSetRecordLocks
- 76 BackoutRecordLocks
- 77 UnenumerateRecordLocks
- 78 ReleaseRecordLocks
- 79 SetVolumeFlags
- 80 ClearVolumeFlags
- 81 GetOriginalInfo
- 82 CreateMigratedDir
- 83 F3OpenCreate
- 84 F3InitFileSearch
- 85 F3ContinueFileSearch
- 86 F3RenameFile
- 87 F3ScanForTrustees
- 88 F3ObtainFileInfo
- 89 F3ModifyInfo
- 90 F3EraseFile
- 91 F3SetDirHandle
- 92 F3AddTrustees
- 93 F3DeleteTrustees
- 94 F3AllocDirHandle
- 95 F3ScanSalvagedFiles
- 96 F3RecoverSalvagedFiles
- 97 F3PurgeSalvageableFile
- 98 F3GetNSSpecificInfo
- 99 F3ModifyNSSpecificInfo
- 100 F3SearchSet
- 101 F3GetDirBase
- 102 F3QueryNameSpaceInfo
- 103 F3GetNamespaceList
- 104 F3GetHugeInfo
- 105 F3SetHugeInfo
- 106 F3GetFullPathString



107 F3GetEffectiveDirectoryRights

108 ParseTree



# 43 Structures

This section describes the Statistical structures and their fields.

# acctngInfo

Contains the accounting information.

## Syntax

```
struct acctngInfo
{
    long    holdTime;
    long    holdAmount;
    long    chargeAmount;
    long    heldConnectTimeInMinutes;
    long    heldRequest;
    byte    heldBytesRead[6];
    byte    heldBytesWritten[6];
};
```

# authInfo

Contains the authentication information.

## Syntax

```
struct authInfo
{
    long    loginStatus;
    long    loginPrivileges;
};
```

## Fields

*loginStatus*

Specifies the current connection's status:

Number	Constant
0x00000001	LOGGED_IN
0x00000002	BEING_ABORTED
0x00000010	MAC_STATION
0x00000020	AUTHENTICATED_TEMPORARY
0x00000100	LOGOUT_IN_PROGRESS
0x00000200	INTERNAL_LOGIN
0x00000400	BINDERY_CONNECTION

# BoardName

Contains the LAN name information.

## Syntax

Offset	Content	Type
0+	DriverBoardName string	byte[]
xx+	DriverShortName string	byte[]
yy+	DriverLogicalName string	byte[]

## Fields

### *DriverBoardName*

Specifies the LAN driver full name that is bound to the board.

### *DriverShortName*

Specifies the LAN driver short name that is bound to the board.

### *DriverLogicalName*

Specifies the LAN driver logical name that is bound to the board.

## Remarks

*DriverBoardName*, *DriverShortName*, and *DriverLogicalName* are NULL-terminated strings. If names are present, a NULL byte will replace these fields.

The *DriverLogicalName* is specified when you load the driver and use a "name=" string on the command line.

# CacheInfo

Contains the cache buffer count information.

## Syntax

Offset	Content	Type
0	MaximumByteCount	long (Lo-Hi)
4	MinimumNumberOfCacheBuffers	long (Lo-Hi)
8	MinimumCacheReportThreshold	long (Lo-Hi)
12	AllocateWaitingCount	long (Lo-Hi)
16	NDirtyBlocks	long (Lo-Hi)
20	CacheDirtyWaitTime	long (Lo-Hi)
24	CacheMaximumConcurrentWrites	long (Lo-Hi)
28	MaximumDirtyTime	long (Lo-Hi)
32	NumberOfDirectoryCacheBuffers	long (Lo-Hi)
36	CacheByteToBlockShiftFactor	long (Lo-Hi)

## Fields

### *MaximumByteCount*

Specifies the maximum length (in bytes) of a cache block.

### *MinimumNumberOfCacheBuffers*

Specifies the minimum number of cache buffers allowed on the server.

### *MinimumCacheReportThreshold*

Specifies the number of cache buffers used for the report threshold.

### *AllocateWaitingCount*

Specifies the number of processes waiting to allocate a cache block.

### *NDirtyBlocks*

Specifies the number of dirty blocks waiting to write to disk.

### *CacheDirtyWaitTime*

Specifies the maximum wait before a write request is written to disk.

### *CacheMaximumConcurrentWrites*

Specifies the maximum number of write requests for changed file data that can be put in the elevator before the disk head begins a sweep across the disk.

### *MaximumDirtyTime*

Specifies the longest time (in ticks) since the server was brought up that a dirty block has waited before it was written to disk.

### *NumberOfDirectoryCacheBuffers*

Specifies the number of directory cache buffers on the server.

### *CacheBytesToBlockShiftFactor*

Specifies the factor used to determine the block size.

## Remarks

The following values can be set using the SET console command:

Field	Supported Values	Default Value
<i>MinimumNumberOfCacheBuffers</i>	20 to 1000	20
<i>MinimumCacheReportThreshold</i>	0 to 1000	20
<i>CacheDirtyWaitTime</i>	0.1 to 10 seconds	3.3 seconds
<i>CacheMaximumConcurrentWrites</i>	10 to 100	50

When the cache buffers reach a number equal to the minimum number of cache buffers plus *MinimumCacheReportThreshold*, the server sends a message warning that cache buffers are getting low.

The block size for *CacheBytesToBlockShiftFactor* is calculated by the following equation, where n is the shift factor: block size =  $2^{n+9}$  bytes.



# CommonLanStruc

Contains information about the common counters for a LAN board.

## Syntax

Offset	Content	Type
0	NotSupportedMask	(Lo-Hi)
4	TotalTxPacketCount	(Lo-Hi)
8	TotalRxPacketCount	(Lo-Hi)
12	NoECBAvailableCount	(Lo-Hi)
16	PacketTxTooBigCount	(Lo-Hi)
20	PacketTxTooSmallCount	(Lo-Hi)
24	PacketRxOverflowCount	(Lo-Hi)
28	PacketRxTooBigCount	(Lo-Hi)
32	PacketRxTooSmallCount	(Lo-Hi)
36	PacketTxMiscErrorCount	(Lo-Hi)
40	PacketRxMiscErrorCount	(Lo-Hi)
44	RetryTxCount	(Lo-Hi)
48	ChecksumErrorCount	(Lo-Hi)
52	HardwareRxMismatchCount	(Lo-Hi)

## Fields

### *NotSupportedMask*

Specifies the fields in the statistical table:

- 0 Bit counter is supported
- 1 Bit counter is not supported

### *TotalTxPacketCount*

Specifies the total number of packets that were transmitted by the LAN board.

### *TotalRxPacketCount*

Specifies the total number of packets that were received by the LAN board.

### *NoECBAvailableCount*

Specifies the number of times the LAN board failed to get a receive ECB.

*PacketTxTooBigCount*

Specifies the number of times the send packet was too big for this LAN board to send.

*PacketTxTooSmallCount*

Specifies the number of times the send packet was too small for this LAN board to send.

*PacketRxOverflowCount*

Specifies the number of times the LAN board's receive buffers overflowed.

*PacketRxTooBigCount*

Specifies the number of times the LAN board could not receive a packet because the packet was too big.

*PacketRxTooSmallCount*

Specifies the number of times the LAN board could not receive a packet because the packet was too small.

*PacketTxMiscErrorCount*

Specifies the number of times any kind of transmit error occurred for the LAN board.

*PacketRxMiscErrorCount*

Specifies the number of times any kind of receive error occurred for the LAN board.

*RetryTxCount*

Specifies the number of times the LAN board retried a transmit because of a previous failure.

*ChecksumErrorCount*

Specifies the number of times a checksum error occurred for the LAN board.

*HardwareRxMismatchCount*

Specifies the counter that is incremented when a packet is received that does not pass length consistency checks (used only by the Ethernet TSU).

# Counters

Contains the cache buffer count information.

## Syntax

Offset	Content	Type
0	ReadExistingBlockCount	long (Lo-Hi)
4	ReadExistingWriteWaitCount	long (Lo-Hi)
8	ReadExistingPartialReadCount	long (Lo-Hi)
12	ReadExistingReadErrorCount	long (Lo-Hi)
16	WriteBlockCount	long (Lo-Hi)
20	WriteEntireBlockCount	long (Lo-Hi)
24	InternalDiskGetCount	long (Lo-Hi)
28	InternalDiskGetNeedToAllocateCount	long (Lo-Hi)
32	InternalDiskGetSomeoneBeatMyCount	long (Lo-Hi)
36	InternalDiskGetPartialReadCount	long (Lo-Hi)
40	InternalDiskGetReadErrorCount	long (Lo-Hi)
44	AsyncInternalDiskGetCount	long (Lo-Hi)
48	AsyncInternalDiskGetNeedToAlloc	long (Lo-Hi)
52	AsyncInternalDiskGetSomeoneBeatMe	long (Lo-Hi)
56	ErrorrDoingAsyncReadCount	long (Lo-Hi)
60	InternalDiskGetNoReadCount	long (Lo-Hi)
64	InternalDiskGetNoReadAllocateCount	long (Lo-Hi)
68	InternalDiskGetNoReadSomeoneBeatMeCount	long (Lo-Hi)
72	InternalDiskWriteCount	long (Lo-Hi)
76	InternalDiskWriteAllocateCount	long (Lo-Hi)
80	InternalDiskWriteSomeoneBeatMeCount	long (Lo-Hi)
84	WriteErrorCount	long (Lo-Hi)
88	WaitOnSemaphoreCount	long (Lo-Hi)
92	AllocateBlockIHadToWaitForSomeoneCount	long (Lo-Hi)
96	AllocateBlockCount	long (Lo-Hi)
100	AllocateBlockIHadToWaitCount	long (Lo-Hi)

# CPUInformation

Contains information about a workstation's CPU.

## Syntax

Offset	Content	Type
0	PageTableOwnerFlag	long (Lo-Hi)
4	CPUType	long (Lo-Hi)
8	CoprocessorFlag	long (Lo-Hi)
12	BusType	long (Lo-Hi)
16	IOEngineFlag	long (Lo-Hi)
20	FSEngineFlag	long (Lo-Hi)
24	NonDedFlag	long (Lo-Hi)
28+	CPUString	byte[]
xx+	Numeric Coprocessor Present/Not Present String	byte[]
yy+	Bus Strings	byte[]

## Fields

### *PageTableOwnerFlag*

Specifies who owns the page table.

### *CPUType*

Specifies the CPU type.

### *CoprocessorFlag*

Specifies whether a coprocessor is present.

### *BusType*

Specifies the type of bus used:

- 0x00 ISA
- 0x01 Micro Channel
- 0x02 EISA
- 0x04 PCI
- 0x08 PCMCIA (Credit Card)
- 0x10 ISA

### *IOEngineFlag*

Specifies whether the IO engine is installed:

TRUE installed  
FALSE Not installed

*FSEngineFlag*

Specifies whether the file system engine is installed:

TRUE installed  
FALSE Not installed

*NonDedFlag*

Specifies whether the CPU is dedicated to the NetWare OS.

*CPUString*

Specifies the CPU string:

"Pentium Pro/0"  
"Pentium/0"  
"80486/0"  
"80386/0"  
"CPU TYPE UNKNOWN/0"

## Remarks

There may be multiple NULL-terminated bus strings that are returned, depending on the machine's hardware configuration.

*NonDedFlag* is set only when the CPU is nondedicated (currently only OS/2 is considered to be nondedicated).

*CPUString*, NPU present string, and the bus strings are in ASCIIZ format.

# CustomCntsInfo

Contains the custom counters information for a LAN board.

## Syntax

Offset	Content
0	CustomVariableValue
4	CustomStringLength
5	CustomString

## Fields

*CustomVariableValue*

Specifies the value of the custom counter.

*CustomStringLength*

Specifies the length of the string that starts with `stringStart`.

*CustomString*

Specifies the first byte of a string that describes the custom counter.

## Remarks

If *CustomStringLength* is zero, there is no string following.

# DirCacheInfo

Contains the directory cache information.

## Syntax

Offset	Content	Type
0	MinimumTimeSinceFileDelete	long (Lo-Hi)
4	AbsMinimumtimeSinceFileDelete	long (Lo-Hi)
8	MinimumNumberOfDirCacheBuffers	long (Lo-Hi)
12	MaximumNumberOfDirCacheBuffers	long (Lo-Hi)
16	NumberOfDirectoryCacheBuffers	long (Lo-Hi)
20	DCMinimumNonReferencedTime	long (Lo-Hi)
24	DCWaitTimeBeforeNewBuffer	long (Lo-Hi)
28	DCMaximumConcurrentWrites	long (Lo-Hi)
32	DCDirtyWaitTime	long (Lo-Hi)
36	DCDoubleReadFlag	long (Lo-Hi)
40	MapHashNodeCount	long (Lo-Hi)
44	SpaceRestrictionNodeCount	long (Lo-Hi)
48	TrusteeListNodeCount	long (Lo-Hi)
52	PercentOfVolumeUsedByDirs	long (Lo-Hi)

## Fields

### *MinimumTimeSinceFileDelete*

Specifies the minimum time (in ticks) between the deletion of a file and when it can be purged.

### *AbsMinimumTimeSinceFileDelete*

Specifies the minimum time (in ticks) between the deletion of a file and when it can be purged when the system has no available blocks.

### *MinimumNumberOfDirCacheBuffers*

Specifies the minimum number of directory cache buffers that can be allocated on the server.

### *MaximumNumberOfDirCacheBuffers*

Specifies the maximum number of directory cache buffers that can be allocated on the server.

### *NumberOfDirectoryCacheBuffers*

Specifies the current number of directory cache buffers on the server.

### *DCMinimumNonReferencedTime*

Specifies the time (in ticks) that must elapse between the last reference of a directory buffer and the time it is reused.

### *DCWaitTimeBeforeNewBuffer*

Specifies the time (in ticks) that must elapse before an additional directory cache buffer can be allocated.

### *DCMaximumConcurrentWrites*

Specifies the maximum number of write requests from directory cache buffers that can be put in the elevator before they are written to disk.

### *DCDirtyWaitTime*

Specifies the maximum time (in ticks) that the server can wait before writing dirty cache buffers to disk.

## Remarks

The following values can be set using the SET console command:

Field	Supported Values	Default Value
<i>MinimumNumberOfDirCacheBuffers</i>	10 to 2000	20
<i>MaximumNumberOfDirCacheBuffers</i>	20 to 4000	50
<i>DCMinimumNonReferencedTime</i>	1 second to 5 minutes	5.5 seconds
<i>DCWaitTimeBeforeNewBuffer</i>	0.5 seconds to 2 minutes	2.2 seconds
<i>DCMaximumConcurrentWrites</i>	5 to 50	10
<i>DCDirtyWaitTime</i>	0 to 10 seconds	0.5 seconds



# ExtraCacheCnts

Contains the cache count information that is used for debugging.

## Syntax

Offset	Content	Type
0	InternalDiskGetNoWaitCount	long (Lo-Hi)
4	InternalDiskGetNoWaitNeedToAllocateCount	long (Lo-Hi)
8	InternalDiskGetNoWaitNoBlockCount	long (Lo-Hi)
12	IDGetNoReadNoWaitCount	long (Lo-Hi)
16	IDGetNoReadNoWaitSemaphoredCount	long (Lo-Hi)
20	IDGetNoReadNoWaitNoBufferCount	long (Lo-Hi)
24	IDGetNoReadNoWaitAllocateCount	long (Lo-Hi)
28	IDGetNoReadNoWaitNoAllocCount	long (Lo-Hi)
32	IDGetNoReadNoWaitNoAllocSemaphoredCount	long (Lo-Hi)
36	IDGetNoReadNoWaitNoAllocAllocateCount	long (Lo-Hi)

# FileInfoStruct

Contains the file information about decompressed files.

## Syntax

```
typedef struct
{
    LONG    parentDirectoryEntryNumber;
    LONG    DirectoryEntryNumber;
    LONG    TotalBlocksToDecompress;
    LONG    CurrentBlockBeingDecompressed;
} FileInfoStruct;
```

# FileServerCounters

Contains information about the file server counters.

## Syntax

Offset	Content	Type
0	TooManyHops	long (Lo-Hi)
2	UnknownNetwork	long (Lo-Hi)
4	NoSpaceForService	long (Lo-Hi)
6	NoReceiveBuffers	long (Lo-Hi)
8	NotMyNetwork	long (Lo-Hi)
10	NetBIOSProgatedCount	long (Lo-Hi)
14	TotalPacketsServiced	long (Lo-Hi)
18	TotalPacketsRouted	long (Lo-Hi)

## Fields

### *TooMayHops*

Specifies the number of packets that were discarded because they had passed through more than 16 bridges without reaching their destination.

### *UnknownNetwork*

Specifies the number of packets that were discarded because their destination network was unknown to the server.

### *NoSpaceForService*

Is always set to zero.

### *NoReceiveBuffers*

Specifies the number of times a packet was discarded because there were no buffers to receive it.

### *NotMyNetwork*

Specifies the number of packets received that were not destined for the server.

### *NetBIOSPropagatedCount*

Specifies the number of NetBIOS packets received that were sent forward.

### *TotalPacketsServiced*

Specifies the total packets that were received by the server.

### *TotalPacketsRouted*

Specifies the number of all packets forwarded by the server.

# FileSystemInfo

Contains the NetWare 3.x file system information.

## Syntax

Offset	Content	Type
0	FATMovedCount	(Lo-Hi)
4	FATWriteErrorCount	(Lo-Hi)
8	SomeoneElseDidItCount0	(Lo-Hi)
12	SomeoneElseDidItCount1	(Lo-Hi)
16	SomeoneElseDidItCount2	(Lo-Hi)
20	IRanOutSomeoneElseDidItCount	(Lo-Hi)
24	IRanOutSomeoneElseDidItCount1	(Lo-Hi)
28	IRanOutSomeoneElseDidItCount2	(Lo-Hi)
32	TurboFATBuildScrewedUpCount	(Lo-Hi)
36	ExtraUseCountNodeCount	(Lo-Hi)
40	ExtraExtraUseCountNodeCount	(Lo-Hi)
44	ErrorReadingLastFATCount	(Lo-Hi)
48	SomeoneElseUsingThisFileCount	(Lo-Hi)

## Fields

### *FATMovedCount*

Specifies the number of times the NetWare server OS has moved the location of the FAT.

### *FATWriteErrorCount*

Specifies the number of disk write errors in both the original and mirrored copy of a disk's FAT sector.

### *SomeoneElseDidItCount0*

Specifies this is used internally by the OS.

### *SomeoneElseDidItCount1*

Specifies this is used internally by the OS.

### *SomeoneElseDidItCount2*

Specifies this is used internally by the OS.

*IRanOutSomeoneElseDidItCount0*

Specifies this is used internally by the OS.

*IRanOutSomeoneElseDidItCount1*

Specifies this is used internally by the OS.

*IRanOutSomeoneElseDidItCount2*

Specifies this is used internally by the OS.

*TurboFATBuildScrewedUpCount*

Specifies the number of times the OS tried to allocate a Turbo FAT index but failed.

*ExtraUseCountNodeCount*

Specifies the number of times the OS tried to allocate a use count node for a TTS transaction but failed.

*ExtraExtraUseCountNodeCount*

Specifies the number of times the OS tried to allocate an additional use count node for a TTS transaction but failed.

*ErrorReadingLastFATCount*

Specifies the number of times the OS received an error reading the data in the last FAT.

*SomeoneElseUsingThisFileCount*

Specifies the number of times the OS was reading a file that another process was also reading.

# GenericInfoDef

Contains the object information.

## Syntax

```
struct GenericInfoDef
{
    struct MediaInfoDef    mediaInfo;
    LONG                   mediatype;
    LONG                   cartridgetype;
    LONG                   unitsize;
    LONG                   blocksize;
    LONG                   capacity;
    LONG                   preferredunitsize;
    BYTE                   name[64];
    LONG                   type;
    LONG                   status;
    LONG                   functionmask;
    LONG                   controlmask;
    LONG                   parentcount;
    LONG                   siblingcount;
    LONG                   childcount;
    LONG                   specificinfosize;
    LONG                   objectuniqueid;
    LONG                   mediaslot;
} GenericStruct;
```

# IPXInformation

Contains the counter information for IPX.

## Syntax

Offset	Content	Type
0	<code>IpxSendPacketCount</code>	(Lo-Hi)
4	<code>IpxMalformPacketCount</code>	(Lo-Hi)
6	<code>IpxGetECBRequestCount</code>	(Lo-Hi)
10	<code>IpxGetECBFailCount</code>	(Lo-Hi)
14	<code>IpxAESEventCount</code>	(Lo-Hi)
18	<code>IpxPostponedAESCount</code>	(Lo-Hi)
20	<code>IpxMaxConfiguredSocketCount</code>	(Lo-Hi)
22	<code>IpxMaxOpenSocketCount</code>	(Lo-Hi)
24	<code>IpxOpenSocketFailCount</code>	(Lo-Hi)
26	<code>IpxListenECBCount</code>	(Lo-Hi)
30	<code>IpxECBCancelFailCount</code>	(Lo-Hi)
32	<code>IpxGetLocalTargetFailCount</code>	(Lo-Hi)

## Fields

### *IpxSendPacketCount*

Specifies the number of IPX packets sent by the server.

### *IpxMalformPacketCount*

Specifies the number of IPX that were discarded because they were malformed.

### *IpxGetECBRequestCount*

Specifies the number of ECB requests.

### *IpxGetECBFailCount*

Specifies the number of times an ECB was requested but could not be supplied.

### *IpxAESEventCount*

Specifies the number of AES events that were submitted.

### *IpxPostponedAESCount*

Specifies the number of AES events that could not be scheduled and were placed in a waiting list.

*IpxMaxConfiguredSocketCount*

Specifies the maximum number of sockets that can be open at one time.

*IpxMaxOpenSocketCount*

Specifies the maximum number of sockets that were open at one time since the server was brought up.

*IpxOpenSocketFailCount*

Specifies the number of times a request to open a socket failed.

*IpxListenECBCount*

Specifies the number of ECBs that are listening for a packet.

*IpxECBCancelFailCount*

Specifies the number of ECB listens that were cancelled.

*IpxGetLocalTargetFailCount*

Specifies the number of times that the server failed to find the target.



# KnownRoutes

Contains route information.

## Syntax

Offset	Content	Type
0	NetIDNumber	long (Lo-Hi)
4	HopsToNet	word (Lo-Hi)
6	NetStatus	word (Lo-Hi)
8	TimeToNet	word (Lo-Hi)

## Fields

*NetIDNumber*

Specifies the net ID.

*HopsToNet*

Specifies the hops to the net.

*NetStatus*

Specifies the router status.

*TimeToNet*

Specifies the time to the net.

# KnownServStruc

Contains information about a known server.

## Syntax

Offset	Content	Type
0	ServerAddress	byte[12]
12	HopsToServer	word (Lo-Hi)
14+	ServerName	byte[]

## Fields

### *ServerAddress*

Specifies the node address of the known server.

### *HopsToNet*

Specifies the hops to the server.

### *ServerName*

Specifies the first byte of the server name (NULL-terminated).

# LANConfigInfo

Contains the configuration information for a LAN board.

## Syntax

Offset	Content	Type
0	LANdriverCFG_MajorVersion	(Lo-Hi)
1	LANdriverCFG_MinorVersion	(Lo-Hi)
2	LANdriverNodeAddress	(Lo-Hi)
8	LANdriverModeFlags	(Lo-Hi)
10	LANdriverBoardNumber	(Lo-Hi)
12	LANdriverBoardInstance	(Lo-Hi)
14	LANdriverMaximumSize	(Lo-Hi)
18	LANdriverMaxRecvSize	(Lo-Hi)
22	LANdriverRecvSize	(Lo-Hi)
26	LANdriverCardID	(Lo-Hi)
28	LANdriverMediaID	(Lo-Hi)
30	LANdriverTransportTime	(Lo-Hi)
32	LANdriverSrcRouting	(Lo-Hi)
36	LANdriverLineSpeed	(Lo-Hi)
38	LANdriverReserved	(Lo-Hi)
40	LANdriverMajorVersion	(Lo-Hi)
41	LANdriverMinorVersion	(Lo-Hi)
42	LANdriverFlags	(Lo-Hi)
44	LANdriverSendRetries	(Lo-Hi)
46	LANdriverLink	(Lo-Hi)
50	LANdriverSharingFlags	(Lo-Hi)
52	LANdriverSlot	(Lo-Hi)
54	LANdriverIOPortsAndRanges	(Lo-Hi)
62	LANdriverMemoryDecode0	(Lo-Hi)
66	LANdriverMemoryLength0	(Lo-Hi)
68	LANdriverMemoryDecode1	(Lo-Hi)

Offset	Content	Type
72	LANdriverMemoryLength1	(Lo-Hi)
74	LANdriverInterrupt	(Lo-Hi)
76	LANdriverDMAUsage	(Lo-Hi)
78	LANdriverLogicalName	(Lo-Hi)
96	LANdriverIOReserved	(Lo-Hi)
110	LANdriverCardName	(Lo-Hi)
238	LANdriverShortName	(Lo-Hi)
278	LANdriverMediaType	(Lo-Hi)
318	LANCustomVariablesCount	(Lo-Hi)

## Fields

### *LANdriverCFG\_MajorVersion*

Specifies the Novell defined major version number of the configuration table.

### *LANdriverCFG\_MinorVersion*

Specifies the Novell defined minor version number of the configuration table.

### *LANdriverNodeAddress*

Specifies the node address of the LAN board.

### *LANdriverModeFlags*

Specifies the modes supported by the driver in bits 14 and 15:

- 00 Unspecified format (node is assumed to be in the physical layer's native format)
- 01 Illegal combination (should not occur)
- 10 Canonical address
- 11 Non-canonical address

### *LANdriverBoardNumber*

Specifies the logical board number assigned to the LAN board by the LSL (1-255).

### *LANdriverBoardInstance*

Specifies the number of the physical card that the logical board is using.

### *LANdriverMaximumSize*

Specifies the maximum send or receive packet size (in bytes) that the LAN board can transmit or receive.

### *LANdriverMaxRecvSize*

Specifies the maximum packet size (or best size) in bytes that the LAN board can receive.

### *LANdriverRecvSize*

Specifies the maximum packet size (in bytes) that a protocol stack can send or receive using this board.

### *LANdriverCardID*

Specifies the number assigned to the LAN board by the IMSP.

### *LANdriverMediaID*

Specifies the number identifying the link-level envelope (frame ID) used by the MLID.

### *LANdriverTransportTime*

Specifies the time (in ticks) that it takes for the LAN board to transmit a 576-byte packet.

### *LANdriverSrcRouting*

Specifies the source router information.

### *LANdriverLineSpeed*

Specifies the line speed.

### *LANdriverReserved*

Is reserved for future use (set to zero).

### *LANdriverMajorVersion*

Specifies the major version number of the MLID.

### *LANdriverMinorVersion*

Specifies the minor version number of the MLID.

### *LANdriverFlags*

Specifies a bit map that indicates the architecture supported by MLID:

Bits 9 and 10 Indicate the Support Mechanisms for Multicast Filtering	
00	Group addressing support defaults to that of the Internet medium. For example, for Ethernet it is the Hash Table; for Token Ring it is the Functional Address.
01	Illegal combination (should not occur)
10	Group addressing is supported by a specialized adapter, but the TSM should filter the addresses.
11	Group addressing is supported by a specialized adapter, and TSM checking is not required.

### *LANdriverSendRetries*

Specifies the number of times that the MLID retries send events before aborting the send.

### *LANdriverLink*

Is used by the LSL.

### *LANdriverSharingFlags*

Specifies a bit map that defines the sharing abilities of the MLID.

### *LANdriverSlot*

Specifies the slot number where the LAN board is installed if the board is running in an MCA or EISA environment (or zero).

### *LANdriverIOPortsAndRanges*

Specifies the I/O port information:

---

1st WORD	Primary base I/O port for the LAN board
2nd WORD	Number of I/O ports beginning with the primary base I/O port
3rd WORD	Secondary base I/O port for the LAN board
4th WORD	Number of I/O ports beginning with the secondary base I/O port

---

### *LANdriverMemoryDecode0*

Specifies the absolute primary memory address that the LAN board uses (or zero).

### *LANdriverMemoryLength0*

Specifies the amount of memory (in paragraphs) that the LAN board uses, starting at *LANdriverMemoryDecode0* (or zero).

### *LANdriverMemoryDecode1*

Specifies the absolute secondary memory address that the LAN board uses (or zero).

### *LANdriverMemoryLength1*

Specifies the amount of memory (in paragraphs) that the LAN board uses, starting at *LANdriverMemoryDecode0* (or zero).

### *LANdriverInterrupt*

Specifies the interrupt vector numbers:

FFh Not used

Byte 1 Primary interrupt vector number

Byte 2 Secondary interrupt vector number

### *LANdriverDMAUsage*

Specifies the DMA channels used by the LAN board:

FFh Not used

Byte 1 Primary DMA channel

Byte 2 Secondary DMA channel

### *LANdriverLogicalName*

Specifies the logical name of the LAN driver that was given at load time.

### *LANdriverIOReserved*

Is reserved for the LSL.

### *LANdriverCardName*

Specifies a pointer to a length-preceded, zero-terminated ASCII description string contained in the OSDATA segment.

### *LANdriverShortName*

Specifies a pointer to a length-preceded, zero-terminated ASCII string that describes the LAN board in eight bytes or less.

### *LANdriverMediaType*

Specifies a pointer to a length-preceded, zero-terminated string that describes the MLID's frame type.

## Remarks

If a driver is driving one physical card, *LANdriverBoardInstance* would be 1 for all the logical boards on this card. If a second physical card is added, *LANdriverBoardInstance* would be 2 for all the logical boards using the second physical card.

*LANdriverCardName* is similar to the description string in the definition table. For example, "NE2000 ETHERNET Driver."

The Independent Manufacturer Support Program (IMSP) assigns strings for the frame type. The following types are defined by Novell:

Frame ID	Frame Type	Protocol ID	Description
0	VIRTUAL_LAN	00h	Used where no Frame ID/MAC envelope is necessary
1	LOCALTALK	00h	Apple LocalTalk frame
2	ETHERNET_II	8137h	Ethernet using a DEC Ethernet II envelope
3	ETHERNET_802.2	E0h	Ethernet using an 802.2 envelope
4	TOKEN-RING	E0h	Token-ring (802.5) using an 802.2 envelope
5	ETHERNET_802.3	00h	IPX 802.3 raw encapsulation
6	802.4	N/A	Token-passing bus envelope
7	NOVELL_PCN2	1111h	Novell's IBM PC Network II envelope
8	GNET	E0h	Gateway's GNET frame envelope
9	PRONET-10	N/A	Proteon's proNET I/O frame envelope
10	ETHERNET_SNAP	8137h	Ethernet (802.3) using an 802.2 envelope with SNAP
11	TOKEN-RING_SNAP	8137h	Token-ring (802.5) using an 802.2 envelope with SNAP
12	LANPAC_II	N/A	Racore's frame envelope
13	ISDN	N/A	Integrated Services Digital Network
14	NOVELL_RX-NET	FAh	Novell's ARCNET envelope

Frame ID	Frame Type	Protocol ID	Description
15	IBM_PCN2_802.2	E0h	IBM PCN2 using 802.2 envelope
16	IBM_PCN2_SNAP	8137h	IBM PCN2 using 802.2 with SNAP envelope
17	OMNINET/4	N/A	Corvus's frame envelope
18	3270_COAXA	N/A	Harris Adacom's frame envelope
19	IP	N/A	IP Tunnel frame envelope
20	FDDI_802.2	E0h	FDDI (802.7) using an 802.2 envelope
21	IVDLAN_802.9	N/A	Commtext, Inc.'s frame envelope
22	DATA_CO_OSI	N/A	Dataco's frame envelope
23	FDDI_SNAP	8137h	FDDI (802.7) using 802.2 with a SNAP envelope
24	IBM_SDL_C	N/A	SDLC tunnel envelope
25	PCO_FDDITP	N/A	PC Office frame envelope
26	WAIDNET	N/A	Hypercommunications
27	SLIP	N/A	Novell frame envelope
28	PPP	N/A	Novell frame envelope



# languageInfo

Contains the language/NCP encoding information.

## Syntax

```
struct languageInfo
{
    long    NCPEncodedStringsBits;
    long    codePage;
};
```

# lockInfo

Contains the lock information.

## Syntax

```
struct lockInfo
{
    byte    logicalLockThreshold;
    byte    physicalLockThreshold;
    word    fileLockCount;
    word    recordLockCount;
};
```

# LSLInformation

Contains the LSL information.

## Syntax

Offset	Content	Type
0	RxBuffers	(Lo-Hi)
4	RxBuffers75%	(Lo-Hi)
8	RxBuffersCheckedOutCount	(Lo-Hi)
12	RxBufferSize	(Lo-Hi)
16	MaximumPhysicalPacketSize	(Lo-Hi)
20	LastTimeRxBufferWasAllocated	(Lo-Hi)
24	MaximumNumberOfProtocols	(Lo-Hi)
28	MaximumNumberOfMediaTypes	(Lo-Hi)
32	TotalTXPackets	(Lo-Hi)
36	GetECBBfrs	(Lo-Hi)
40	GetECBFails	(Lo-Hi)
44	AESEventCounts	(Lo-Hi)
48	PostponedEvents	(Lo-Hi)
52	ECBCxIFails	(Lo-Hi)
56	ValidBfrsReused	(Lo-Hi)
60	EnqueuedSendCnt	(Lo-Hi)
64	TotalRXPackets	(Lo-Hi)
68	UnclaimedPackets	(Lo-Hi)
72	StatTableMajorVersion	(Lo-Hi)
73	StatTableMinorVersion	(Lo-Hi)

## Fields

*RxBuffers*

Specifies the total number of LSL receive buffers.

*RxBuffers75%*

Specifies the number of LSL receive buffers that must be in use before a warning message is issued that buffers are getting low.

*RxBuffersCheckedOutCount*

Specifies the number of LSL buffers that are currently in use.

*RxBufferSize*

Specifies the size of the data portion of the ECBs (in bytes).

*MaximumPhysicalPacketSize*

Specifies the total size of the ECB (in bytes).

*LastTimeRxBufferWasAllocated*

Specifies the last time (in ticks since the server was brought up) that a buffer was checked out.

*MaximumNumberOfProtocols*

Specifies the number of protocol stacks supported by the OS.

*MaximumNumberOfMediaTypes*

Specifies the number of frame types supported by the OS.

*TotalTXPackets*

Specifies the number of packet transmit requests.

*GetECBBfrs*

Specifies the number of ECBs that were requested.

*GetECBFails*

Specifies the number of times an ECB failed.

*AESEventCounts*

Specifies the total number of AES events that have been processed.

*PostponedEvents*

Specifies the total number of AES events that were postponed because of critical sections.

*ECBCxlFails*

Specifies the number of AES cancel requests that failed because the event was not found on the AES list.

*ValidBfrsReused*

Specifies the number of ECBs in the hold queue that were reused before they were removed from the hold queue.

*EnqueuedSendCnt*

Specifies the number of send events in the queue that have occurred.

*TotalRXPackets*

Specifies the total number of received incoming packets.

*UnclaimedPackets*

Specifies the total number of unclaimed incoming packets.

*StatTableMajorVersion*

Specifies the major version of the LSL statistics table.

*StatTableMinorVersion*

Specifies the minor version of the LSL statistics table.

# MemoryCounters

Contains the memory count information.

## Syntax

Offset	Content	Type
0	OriginalNumberOfCacheBuffers	long (Lo-Hi)
4	CurrentNumberOfCacheBuffers	long (Lo-Hi)
8	CacheDirtyBlockThreshold	long (Lo-Hi)
12	WaitNodeCount	long (Lo-Hi)
16	WaitNodeAllocFailureCount	long (Lo-Hi)
20	MoveCacheNodeCount	long (Lo-Hi)
24	MoveCacheNodeFromAvailCount	long (Lo-Hi)
28	AccelerateCacheNodeWriteCount	long (Lo-Hi)
32	RemoveCacheNodeCount	long (Lo-Hi)
36	RemoveCacheNodeFromAvailCount	long (Lo-Hi)

## Fields

### *OriginalNumberOfCacheBuffers*

Specifies the number of cache buffers that existed when the server was brought up.

### *CurrentNumberOfCacheBuffers*

Specifies the number of cache buffers that are presently on the server.

### *CacheDirtyBlockThreshold*

Specifies the maximum number of cache blocks that are simultaneously allowed to be dirty.

# MLIDBoardInfo

Contains the MLID board information.

## Syntax

Offset	Content	Type
0	ProtocolBoardNum	long
4	ProtocolNumber	word
6	ProtocolID	byte[6]
7	ProtocolName	byte[16]

## Fields

*ProtocolBoardNum*

Specifies the board number the protocol is using.

*ProtocolNumber*

Specifies the protocol number.

*ProtocolID*

Specifies the protocol ID (fixed-length 6-byte string).

*ProtocolName*

Specifies the protocol name (length-preceded 16-byte string).

# nameInfo

Contains the name information.

## Syntax

```
struct nameInfo
{
    byte    loginObjectType;
    long    loginNameLength;
    byte    loginName[loginNameLength];
};
```



# ncpNetworkAddress

Contains the network address information.

## Syntax

```
struct ncpNetworkAddress
{
    long    addressType;
    long    addressSize;
    byte    address[addressSize];
};
```

# netAddr

Contains the transport information.

## Syntax

```
struct netAddr
{
    byte    transportType;
    byte    transportAddressSize;
    byte    transportAddress[transportAddressSize];
};
```

# NLMInformation

Contains the NLM information.

## Syntax

Offset	Content	Type
0	IdentificationNumber	long
4	Flags	long
8	Type	long
12	ParentID	long
16	MajorVersion	long
20	MinorVersion	long
24	Revision	long
28	Year	long
32	Month	long
36	Day	long
40	AllocAvailByte	long
44	AllocFreeCount	long
48	LastGarbCollect	long
52	MessageLanguage	long
56	NumberOfReferencedPublics	long

## Fields

### *IdentificationNumber*

Specifies the number assigned to the NLM when it was loaded.

### *Flags*

Specifies the following values:

Value	Constant	Description
0x0001	REENTRANT	The module is reentrant. If the NLM is loaded twice, the actual code in the server's memory is reused.
0x0002	MULTIPLE	The module can be loaded more than once by a LOAD console command.

Value	Constant	Description
0x0004	SYNCHRONIZE	The load process will go to sleep until the NLM calls SynchronizeStart. This value prevents other console commands from being processed (particularly LOAD commands) while the NLM is being loaded.
0x0008	PSEUDOPREEMPTION	Enables the OS to force an NLM to relinquish control if the NLM does not do so on its own often enough.

### *Type*

Specifies the NLM type.

### *ParentID*

Specifies the number of the NLM assigned to the NLM that caused this NLM to be loaded.

### *MajorVersion*

Specifies the major version number of the NLM

### *MinorVersion*

Specifies the minor version number of the NLM.

### *Revision*

Specifies the revision letter of the NLM.

### *Year*

Specifies the year the NLM was created.

### *Month*

Specifies the month the NLM was created.

### *Day*

Specifies the day the NLM was created.

### *AllocAvailBytes*

Specifies the bytes available for allocation by the NLM.

### *AllocFreeCount*

Specifies the number of free bytes that can be reclaimed.

### *LastGarbCollect*

Specifies the last time garbage collection was done for the NLM (in ticks since the server was brought up).

### *MessageLanguage*

Specifies the language the NLM uses (as a number).

### *NumberOfReferencedPublics*

Specifies the number of external symbols referenced by the NLM.

## Remarks

*Type* can have the following values:

Number	Extension	Description
1	.LAN	LAN driver
2	.DSK	Disk Driver
3	.NAM	Name space support module
4	.NLM	Utility or support program
5	.MSL	Mirrored Server Link
6	.NLM	OS NLM
7	.NLM	Paged high OS NLM
8	.HAM	Host Adapter Module (works with Custom Device Modules)
9	.CDM	Custom Device Module (works with Host Adapter Modules)
10	.NLM	File system engine
11	.NLM	Read mode NLM
12	.NLM	Hidden NLM

# PacketBurstInformation

Contains the packet burst information.

## Syntax

Offset	Content	Type
0	BigInvalidSlotCount	Lo-Hi
4	BigForgedPacketCount	Lo-Hi
8	BigInvalidPacketCount	Lo-Hi
12	BigStillTransmittingCount	Lo-Hi
16	StillDoingTheLastRequestCount	Lo-Hi
20	InvalidControlRequestCount	Lo-Hi
24	ControlInvalidMessageNumberCount	Lo-Hi
28	ControlBeingTornDownCount	Lo-Hi
32	BigRepeatTheFileReadCount	Lo-Hi
36	BigSendExtraCCCount	Lo-Hi
40	BigReturnAbortMessageCount	Lo-Hi
44	BigReadInvalidMessageNumberCount	Lo-Hi
48	BigReadDoItOverCount	Lo-Hi
52	BigReadBeingTornDownCount	Lo-Hi
56	PreviousControlPacketCount	Lo-Hi
60	SendHoldOffMessageCount	Lo-Hi
64	BigReadNoDataAvailableCount	Lo-Hi
68	BigReadTryingToReadTooMuchCount	Lo-Hi
72	AsyncReadErrorCount	Lo-Hi
76	BigReadPhysicalReadErrorCount	Lo-Hi
80	ControlBadACKFragmentListcount	Lo-Hi
84	ControlNoDataReadCount	Lo-Hi
88	WriteDuplicateRequestCount	Lo-Hi
92	ShouldntBeACKingHereCount	Lo-Hi
96	WriteInconsistentPacketLengthsCount	Lo-Hi
100	FirstPacketIsntAWriteCount	Lo-Hi

Offset	Content	Type
104	WriteTrashedDuplicateRequestCount	Lo-Hi
108	BigWriteInvalidMessageNumberCount	Lo-Hi
112	BigWriteBeingTornDownCount	Lo-Hi
116	BigWriteBeingAbortedCount	Lo-Hi
120	ZeroACKFragmentCountCount	Lo-Hi
124	WriteCurrentlyTransmittingCount	Lo-Hi
128	TryingToWriteTooMuchCount	Lo-Hi
132	WriteOutOfMemoryForControlNodesCount	Lo-Hi
136	WriteDidntNeedThisFragmentCount	Lo-Hi
140	WriteTooManyBuffersCheckedOutCount	Lo-Hi
144	WriteTimeOutCount	Lo-Hi
148	WriteGotAnACKCount	Lo-Hi
152	WriteGotAnACKCount1	Lo-Hi
156	PollerAbortedTheConnnectionCount	Lo-Hi
160	MaybeHadOutOfOrderWritesCount	Lo-Hi
164	HadAnOutOfOrderWriteCount	Lo-Hi
168	MovedTheACKBitDownCount	Lo-Hi
172	BumpedOutOfOrderWriteCount	Lo-Hi
176	PollerRemovedOldOutOfOrderCount	Lo-Hi
180	WriteDidntNeedButRequestedACKCount	Lo-Hi
184	WriteTrashedPacketCount	Lo-Hi
188	TooManyACKFragmentsCount	Lo-Hi
192	SavedAnOutOfOrderPacketCount	Lo-Hi
196	ConnectionBeingAbortedCount	Lo-Hi

# printInfo

Contains the print information.

## Syntax

```
struct printInfo
{
    byte    printFlags;
    byte    printTabSize;
    byte    numberOfPrintCopies;
    byte    printToFileFlag;
    byte    printBanner[14];
    byte    targetPrinter;
    byte    formType;
};
```

## Fields

### *printFlags*

Specifies the print flags:

Value	Constant and Description
0x08	SFormSuppressBit specifies the print service suppresses automatic form feed after the print job is printed.
0x10	SCreateBit
0x20	SDeleteBit
0x40	STabBit
0x80	sBannerPageBit specifies the print service precedes the print job with a banner page.

### *tabSize*

Specifies the tab size which is a value between 1 and 18 inclusive (default setting is 0x08).

### *numberCopies*

Specifies the number of copies (0 to 255) of the captured file that is printed (default setting is 0x0001). If 0x0000, nothing prints.

### *printToFileFlag*

Specifies that the data is sent to a file rather than a printer.

### *bannerFileName*

Specifies the name of the banner that is printed when a print job is submitted.

### *targetServerID*

Specifies the server ID of the queue server servicing the job (as indexed into the SpoolerToQMap table). If this field is set to 0xFFFFFFFF, any queue server can service the



job. If the specified queue server is not attached to the queue, QMS removes the job from the queue.

*formType*

Specifies the type of form (0 to 255) a user must mount in the printer to print files captured to the LPT device. If the form currently mounted in the printer differs from the form type returned in this field, the NetWare server console displays a message instructing the console operator to mount the correct form. The default form is 0x0000.

# RoutersInfo

Contains information about each network router.

## Syntax

Offset	Content	Type
0	Node	byte[6]
6	ConnectedLAN	long (Lo-Hi)
10	RouteHops	word (Lo-Hi)
12	RouteTime	word (Lo-Hi)

## Fields

### *Node*

Specifies the 6-byte network address of the router.

### *ConnectedLAN*

Specifies the LAN board number of the router.

### *RouteHops*

Specifies the number of hops to the specified network.

### *RouteTime*

Specifies the time (in ticks) to the specified network.

# RTagStructure

Contains the information for each resource tag used on the server.

## Syntax

Offset	Content	Type
0	RTagNumber	long (Lo-Hi)
4	ResourceSignature	long (Lo-Hi)
8	ResourceCount	long (Lo-Hi)
12	ResourceName (ASCIIZ)	byte[]

## Fields

- RTagNumber*  
Specifies the number assigned to the resource tag when it was allocated.
- ResourceSignature*  
Specifies the signature of the resource tag.
- ResourceCount*  
Specifies the number of this kind of tag that have been allocated.
- ResourceName*  
Specifies the name of the resource tag.

# Segments

Contains information about the server's volume segments.

## Syntax

Offset	Content	Type
0	VolumeSegmentDeviceNumber	long (Lo-Hi)
4	VolumeSegmentOffset	long (Lo-Hi)
8	VolumeSegmentSize	long (Lo-Hi)

## Fields

*VolumeSegmentDeviceNumber*

Specifies the device that the segment is located on.

*VolumeSegmentOffset*

Specifies the offset of the segment (in bytes).

*VolumeSegmentSize*

Specifies the segment size (in bytes).

# ServerInfo

Contains the server information.

## Syntax

Offset	Content	Type
0	ReplyCanceledCount	long (Lo-Hi)
4	WriteHeldOffCount	long (Lo-Hi)
8	WriteHeldOffWithDuplicateRequest	long (Lo-Hi)
12	InvalidRequestTypeCount	long (Lo-Hi)
16	BeingAbortedCount	long (Lo-Hi)
20	AlreadyDoingReAllocateCount	long (Lo-Hi)
24	DeAllocateInvalidSlotCount	long (Lo-Hi)
28	DeAllocateBeingProcessedCount	long (Lo-Hi)
32	DeAllocateForgedPacketCount	long (Lo-Hi)
36	DeAllocateStillTransmittingCount	long (Lo-Hi)
40	StartStationErrorCount	long (Lo-Hi)
44	InvalidSlotCount	long (Lo-Hi)
48	BeingProcessedCount	long (Lo-Hi)
52	ForgedPacketCount	long (Lo-Hi)
56	StillTransmittingCount	long (Lo-Hi)
60	ReExecuteRequestCount	long (Lo-Hi)
64	InvalidSequenceNumberCount	long (Lo-Hi)
68	DuplicateIsBeingSentAlreadyCount	long (Lo-Hi)
72	SentPositiveAcknowledgeCount	long (Lo-Hi)
76	SentADuplicateReplyCount	long (Lo-Hi)
80	NoMemoryForStationControlCount	long (Lo-Hi)
84	NoAvailableConnectionsCount	long (Lo-Hi)
88	ReAllocateSlotCount	long (Lo-Hi)
92	ReAllocateSlotCameTooSoonCount	long (Lo-Hi)

## Fields

### *ReplyCanceledCount*

Specifies the number of replies that were cancelled because the connection was reallocated while the request was being processed.

### *WriteHeldOffCount*

Specifies the number of times that writes were delayed because of a pending TTS transaction or cache busy condition.

### *InvalidRequestTypeCount*

Specifies the number of packets that were received that had an invalid request type or were received after the server was downed.

### *BeingAbortedCount*

Specifies the number of packets received for a connection that was being terminated.

### *AlreadyDoingReAllocateCount*

Specifies the number of times that a connection is requested when a connection already exists.

### *DeAllocateInvalidSlotCount*

Specifies the number of times an attempt was made to deallocate a connection slot that was not valid.

### *StartStationErrorCount*

Specifies the number of times the server was unable to allocate a connection for any reason.

### *InvalidSlotCount*

Specifies the number of requests received for an invalid connection slot.

### *BeingProcessedCount*

Specifies the number of times a duplicate request was received while processing the first request.

### *ForgedPacketCount*

Specifies the number of suspicious invalid packets that were received.

### *StillTransmittingCount*

Specifies the number of times a new request is received before a reply to a previous request has been sent.

### *ReExecuteRequestCount*

Specifies the number of times the requester did not receive a reply and the request had to be reprocessed.

### *InvalidSequenceNumberCount*

Specifies the number of request packets the server received from a connection where the sequence number in the packet did not match the current sequence number or the next sequence number.

*DuplicateIsBeingSentAlreadyCount*

Specifies the number of times a duplicate reply was requested when the reply had already been sent.

*SentPositiveAcknowledgeCount*

Specifies the number of acknowledgments (indication that a connection has repeated a request that is being serviced) that were sent by the server.

*SentADuplicateReplyCount*

Specifies the number of packets for which the server had to send a duplicate reply (request that server cannot process).

*NoMemoryForStationControlCount*

Specifies the number of times that the server could not allocate memory to expand the connection table for a new connection.

*NoAvailableConnectionsCount*

Specifies the number of times there were no slots available in the connection table for a new connection.

*ReAllocateSlotCount*

Specifies the number of times the server reallocated the same slot in the connection table for a client that logged out and then re-logged in.

*ReAllocateSlotCameTooSoonCount*

Specifies the number of times that a request came from a client to re-log in before that client had been completely logged out.

## Remarks

It is rare that suspicious packets can be created due to faulty equipment. If *ForgedPacketCount* is large, it may indicate an attempt to breach network security.

Packets with bad sequence numbers are discarded. If *InvalidSequenceNumberCount* is large, it may indicate an attempt to breach network security.

# ServersSrcInfo

Contains server source information.

## Syntax

Offset	Content	Type
0	ServerNode	byte[6]
6	ConnectedLAN	long (Hi-Lo)
10	SourceHops	word (Hi-Lo)

## Fields

### *ServerNode*

Specifies the node address of a server.

### *ConnectedLAN*

Specifies the LAN board number of the server.

### *SourceHops*

Specifies the number of hops to the server.



# SPXInformation

Contains the counter information for SPX.

## Syntax

Offset	Content	Type
0	SpxMaxConnectionsCount	(Lo-Hi)
2	SpxMaxUsedConnections	(Lo-Hi)
4	SpxEstConnectionReq	(Lo-Hi)
6	SpxEstConnectionFail	(Lo-Hi)
8	SpxListenConnectReq	(Lo-Hi)
10	SpxListenConnectFail	(Lo-Hi)
12	SpxSendCount	(Lo-Hi)
16	SpxWindowChokeCount	(Lo-Hi)
20	SpxBadSendCount	(Lo-Hi)
22	SpxSendFailCount	(Lo-Hi)
24	SpxAbortedConnection	(Lo-Hi)
26	SpxListenPacketCount	(Lo-Hi)
30	SpxBadListenCount	(Lo-Hi)
32	SpxIncomingPacketCount	(Lo-Hi)
36	SpxBadInPacketCnt	(Lo-Hi)
38	SpxSuppressedPackCnt	(Lo-Hi)
40	SpxNoSesListenECBCnt	(Lo-Hi)
42	SpxWatchDogDestSesCnt	(Lo-Hi)

## Fields

*SpxMaxConnectionsCount*

Specifies the maximum number of SPX connections that are allowed on the server.

*SpxMaxUsedConnections*

Specifies the maximum number of SPX connections that were used at one time since the server was brought up.

*SpxEstConnectionReq*

Specifies the total number of SPX connections that were established since the server was brought up.

*SpxEstConnectionFail*

Specifies the number of times since the server was brought up that an attempt to establish an SPX connection failed.

*SpxListenConnectReq*

Specifies the number of requests that were received since the server was brought up to post a listen.

*SpxListenConnectFail*

Specifies the number of times since the server was brought up that a request to post a list item failed.

*SpxSendCount*

Specifies the number of SPX packets sent since the server was brought up.

*SpxWindowChokeCount*

Specifies the internal value used for debugging.

*SpxBadSendCount*

Specifies the number of bad packets that were sent since the server was brought up.

*SpxSendFailCount*

Specifies the number of packets that were sent since the server was brought up for which there was no acknowledgment received.

*SpxAbortedConnection*

Specifies the number of times since the server was brought up that a connection was aborted.

*SpxListenPacketCount*

Specifies the number of times since the server was brought up that a listen was posted on a socket.

*SpxBadListenCount*

Specifies the number of times since the server was brought up that a listen on a socket failed for any reason.

*SpxIncomingPacketCount*

Specifies the number of packets in the queue.

*SpxBadInPacketCnt*

Specifies the number of bad SPX packets that were received since the server was brought up.

*SpxSuppressedPackCnt*

Specifies the number of times a duplicate SPX packet was received.

*SpxNoSesListenECBCnt*

Specifies the number of times since the server was brought up that a listen was posted on a session that was not established.

*SpxWatchDogDestSesCnt*

Specifies the number of times since the server was brought up that the watchdog destroyed a session.

# StackInfo

Contains information about the stack.

## Syntax

```
struct
{
    LONG    StackNumber; (Lo-Hi)
    BYTE    StackShortName[16];
};
```

## Fields

*StackNumber*

Specifies the protocol number.

*StackShortName*

Specifies the protocol name.

## statsInfo

Contains the statistical information.

### Syntax

```
struct statsInfo
{
    byte    totalBytesRead[6];
    byte    totalBytesWritten[6];
    long    totalRequests;
};
```

# timeInfo

Contains the time information.

## Syntax

```
struct timeInfo
{
    byte    loginTime[7];
    long    loginExpirationTime;
};
```

# TrendCounters

Contains the trend cache buffer count information.

## Syntax

Offset	Content	Type
0	NumOfCacheChecks	long (Lo-Hi)
4	NumOfCacheHits	long (Lo-Hi)
8	NumOfDirtyCacheChecks	long (Lo-Hi)
12	NumOfCacheDirtyChecks	long (Lo-Hi)
16	CacheUsedWhileChecking	long (Lo-Hi)
20	WaitTillDirtyBlocksDecreaseCnt	long (Lo-Hi)
24	AllocateBlockFromAvailCount	long (Lo-Hi)
28	AllocateBlockFromLRUCount	long (Lo-Hi)
32	AllocateBlockAlreadyWaiting	long (Lo-Hi)
36	LRUSittingTime	long (Lo-Hi)
40	NumOfCacheCheckNoWait	long (Lo-Hi)
44	NumOfCacheHitsNoWait	long (Lo-Hi)

## Fields

### *NumOfCacheChecks*

Specifies the total number of times any block in the cache was looked at since the server was brought up.

### *NumberOfCacheHits*

Specifies the number of times cache requests were serviced from existing cache blocks.

### *LRUSittingTime*

Specifies the time in ticks that the oldest cache block has been waiting (sitting in the LRU list).

# UserInformation

Contains information about the user.

## Syntax

Offset	Content	Type
0	connectionNumber	Lo-Hi
4	useCount	Lo-Hi
8	connectionServiceType	Lo-Hi
9	loginTime	Lo-Hi
16	status	Lo-Hi
20	expirationTime	Lo-Hi
24	objectType	Lo-Hi
28	transactionFlag	Lo-Hi
30	logicalLockThreshold	Lo-Hi
31	fileWriteFlags	Lo-Hi
32	fileWriteState	Lo-Hi
33	filler	Lo-Hi
34	fileLockCount	Lo-Hi
36	recordLockCount	Lo-Hi
38	totalBytesRead	Lo-Hi
44	totalBytesWritten	Lo-Hi
50	totalRequests	Lo-Hi
54	heldRequests	Lo-Hi
58	heldBytesRead	Lo-Hi
64	heldBytesWritten	Lo-Hi

## Fields

*connectionNumber*

Specifies the connection number of the user.

*useCount*

Specifies whether the connection is in use:

0 Connection is not in use



1 Connection is in use

*connectionServiceType*

Specifies the connection type:

- 1 Included for CLIB backward compatibility
- 2 NCP\_CONNECTION\_TYPE
- 3 NLM\_CONNECTION\_TYPE
- 4 AFP\_CONNECTION\_TYPE
- 5 FTAM\_CONNECTION\_TYPE
- 6 ANCP\_CONNECTION\_TYPE
- 7 ACP\_CONNECTION\_TYPE
- 8 SMB\_CONNECTION\_TYPE
- 9 WINSOCK\_CONNECTION\_TYPE

*loginTime*

Specifies the time that the user logged in.

*status*

Specifies the status of the connection:

- 0x00000001 LOGGED\_IN
- 0x00000002 BEING\_ABORTED
- 0x00000004 AUDITED
- 0x00000008 NEEDS\_SECURITY\_CHANGE
- 0x00000010 MAC\_STATION
- 0x00000020 AUTHENTICATED\_TEMPORARY
- 0x00000040 AUDIT\_CONNECTION\_RECORDED
- 0x00000080 DSAUDIT\_CONNECTION\_RECORDED

*expirationTime*

Specifies the time before the authentication expires (in ticks).

*objectType*

Specifies the type of the user.

*transactionFlag*

Specifies the transaction tracking information.

*logicalLockThreshold*

Specifies the maximum number of logical locks the user can have.

*recordLockThreshold*

Specifies the maximum number of record locks the user can have.

*fileWriteFlags*

Specifies the writing status:

- 1 Writing
- 2 Write aborted

*fileWriteState*

Specifies the writing status:

- 0 Not writing
- 1 Write in progress
- 2 Write being stopped

*filler*

Is reserved for future use.

*fileLockCount*

Specifies the number of files the user has locked.

*recordLockCount*

Specifies the number of records the user has locked.

*totalBytesRead*

Specifies the number of bytes the user has read (48-bit value).

*totalBytesWritten*

Specifies the number of bytes the user has written (48-bit value).

*totalRequests*

Specifies the number of requests the user has sent.

*heldRequests*

Specifies the number of requests held for accounting purposes.

*heldBytesRead*

Specifies the number of bytes the user has read that have a hold on them for accounting purposes.

*heldBytesWritten*

Specifies the number of bytes the user has written that have a hold on them for accounting purposes.

## Remarks

An *expirationTime* of zero indicates there is no expiration time.

# VolInfoDef

Contains the volume information (*InfoLevelNumber* = 1).

## Syntax

```
struct VolInfoDef
{
    long    VolumeType;
    long    StatusFlagBits;
    long    SectorSize;
    long    SectorsPerCluster;
    long    VolumeSizeInClusters;
    long    FreedClusters;
    long    SubAllocFreeableClusters;
    long    FreeableLimboSectors;
    long    NonFreeableLimboSectors;
    long    NonFreeableAvailableSubAllocSectors;
    long    NotUsableSubAllocSectors;
    long    SubAllocClusters;
    long    DataStreamsCount;
    long    LimboDataStreamsCount;
    long    OldestDeletedFileAgeInTicks;
    long    CompressedDataStreamsCount;
    long    CompressedLimboDataStreamsCount;
    long    UncompressableDataStreamsCount;
    long    PreCompressedSectors;
    long    CompressedSectors;
    long    MigratedFiles;
    long    MigratedSectors;
    long    ClustersUsedByFAT;
    long    ClustersUsedByDirectories;
    long    ClustersUsedByExtendedDirectories;
    long    TotalDirectoryEntries;
    long    UnusedDirectoryEntries;
    long    TotalExtendedDirectoryExtants;
    long    UnusedExtendedDirectoryExtants;
    long    ExtendedAttributesDefined;
    long    ExtendedAttributeExtantsUsed;
    long    DirectoryServicesObjectID;
    long    VolumeLastModifiedDateAndTime;
};
```

## Fields

### *VolumeType*

Specifies the defined type of the current volume:

- 0 VINetWare386
- 1 VINetWare286
- 2 VINetWare386x30
- 3 VINetWare386v31

### *StatusFlagBits*

Specifies the options that are currently available on the volume:

0x01 SubAllocEnableBit  
0x02 CompressionEnabledBit  
0x04 MigrationEnableBit  
0x08 AuditingEnabledBit  
0x10 ReadOnlyEnableBit  
0x20 ImmediatePurgeBit  
0x80000000 NSSVolumeBit

*SectorSize*

Specifies the sector size (in bytes).

*SectorsPerCluster*

Specifies the number of sectors per cluster or block.

*VolumeSizeInClusters*

Specifies the size of the volume (in clusters or blocks).

*FreedClusters*

Specifies the number of clusters or blocks that are currently free for allocation (does not include space that is currently available from deleted or limbo files, nor space that could be reclaimed from the suballocation file system).

*SubAllocFreeableClusters*

Specifies the space that can be reclaimed from the suballocation file system.

*FreeableLimboSectors*

Specifies the disk space (in clusters or blocks) that can be freed from deleted files.

*NonFreeableLimboSectors*

Specifies the disk space (in clusters or blocks) that is currently in deleted files and is not aged enough to be classified as *freeableLimboClusters*.

*NonFreeableAvailSubAllocSectors*

Specifies the space available to the suballocation file system, but not freeable to return as clusters or blocks.

*NotUsableSubAllocSectors*

Specifies the disk space that is wasted by the suballocation file system. These clusters cannot be allocated by the suballocation system or used as regular clusters or blocks.

*SubAllocClusters*

Specifies the disk space being used by the suballocation file system.

*DataStreamsCount*

Specifies the number of data streams for real files that have data allocated to them.

*LimboDataStreamsCount*

Specifies the number of data streams for deleted files that have data allocated to them.

*OldestDeletedFileAgeInTicks*

Specifies the current age of the oldest file (in ticks).

*CompressedDataStreamsCount*

Specifies the number of data streams for compressed real files.

*CompressedLimboDataStreamsCount*

Specifies the number of data streams for compressed deleted files.

*UnCompressableDataStreamsCount*

Specifies the number of data streams that are not compressable (real and deleted).

*PreCompressedSectors*

Specifies the amount of disk space that was allocated to all files before they were compressed (includes "hole" space).

*CompressedSectors*

Specifies the amount of disk space that is used by all compressed files.

*MigratedFiles*

Specifies the number of migrated files.

*MigratedSectors*

Specifies the amount of migrated disk space (in sectors).

*ClustersUsedByFAT*

Specifies the amount of disk space (in clusters or blocks) being used by the FAT table.

*ClustersUsedByDirectories*

Specifies the amount of disk space (in clusters or blocks) being used by directories.

*ClustersUsedbyExtendedDirs*

Specifies the amount of disk space (in clusters or blocks) being used by the extended directory space.

*TotalDirectoryEntries*

Specifies the total number of directories that are available on the volume.

*UnusedDirectoryEntries*

Specifies the total number of directory entries that are not in use on the volume.

*TotalExtendedDirectoryExtants*

Specifies the amount of extended directory space extants (128 bytes each) that are available on the volume.

*UnusedExtendedDirectoryExtants*

Specifies the amount of extended directory space extants (128 bytes each) that are not in use on the volume.

*ExtendedAttributesDefined*

Specifies the number of extended attributes that are defined on the volume.

*ExtendedAttributeExtantsUsed*

Specifies the number of extended directory extants that are used by the extended attributes.

*DirectoryServicesObjectID*

Specifies the NDS ID for the volume.

*VolumeLastModifiedDateAndTime*

Specifies the last time any file or subdirectory on the volume was modified (as tracked by the OS).

# VolInfo2Def

Contains the volume information (*InfoLevelNumber* = 2).

## Syntax

```
struct VolInfo2Def
{
    long    VolumeActiveCount;
    long    VolumeUseCount;
    long    MACRootIDs;
    long    VolumeLastModifiedDateAndTime;
    long    VolumeReferenceCount;
    long    CompressionLowerLimit;
    long    OutstandingIOs;
    long    OutstandingCompressionIOs;
    long    CompressionIOsLimit;
};
```

## Fields

### *VolumeActiveCount*

Specifies a per-volume count that indicates the volume currently has operations in progress that need to be completed before other actions (such as dismounting a volume) can be initiated.

### *VolumeUseCount*

Specifies a per-volume count that is incremented before directory cache requests and other writes to the directory can begin. This field is decremented when such operations complete.

### *MACRootIDs*

Specifies a the DirectoryNumbers of the Macintosh roots.

### *VolumeLastModifiedDateAndTime*

Specifies the last date and time that any subdirectory on the volume was modified.

### *VolumeReferenceCount*

Specifies the last modified reference count. This field is increment every time that MarkDirectoryChanged is called on the volume.

### *CompressionLowerLimit*

Specifies a value used to determine if a file will be smaller once it is compressed. On a volume with suballocation enabled, this value is set to 512; otherwise, it is determined (in part) by the defined block size of the volume.

### *OutstandingIOs*

Specifies the number of IO operations that are still in progress and have not yet completed.

### *OutstandingCompressionIOs*

Specifies the number of compression-related IO operations that are still in progress and have not yet completed.

### *CompressionIOsLimit*

Specifies a per-volume count of the number of IO operations that can be dedicated to compression as opposed to the number of regular reading and writing operations.



# XIX Synchronization

Synchronization NCPs enable you to coordinate access to network files and other network resources. They are divided into two general categories: File and Record Lock requests and Semaphore requests.

To access Synchronization NCPs, use the following:

- ♦ [Chapter 44, “Concepts,” on page 1159](#)
- ♦ [Chapter 45, “NCPs,” on page 1161](#)
- ♦ [Chapter 46, “Structures,” on page 1213](#)



# 44 Concepts

This section explains ideas that are common to Synchronization NCPs.

## File and Record Locks

NCPs support file locking, physical record locking, and logical record locking. A file lock locks a complete file; a physical record lock locks a specified byte range within a file; a logical record lock locks a logical record name associated with one or more files and records.

All files and records must first be logged in to be locked. Logging files and records before locking them ensures that either all necessary files or records are locked or none are locked. When you log files or records, information (such as location, size, and byte range) is recorded in a log table on the file server. This log table is exclusively associated with the requesting task on the workstation.

The requests that log files and records also allow you to immediately lock them. If immediate locking is specified in a login request, you can specify a timeout period that the server uses to retry locking. If files or records are not logged for immediate locking, other requests are provided for locking them. If all logged files or records are available, the file server locks them when the locks are requested. If, however, any of the files or records are being used by another client and the timeout value has expired, the lock request fails.

Unlike a physical record lock, a logical record lock does not actually lock a byte range in a file. Instead, a logical record lock acts somewhat like a semaphore. You cooperatively define a logical record name that represents a group of files, records, or structures. In this case, you lock only the logical record name and not the actual group of files, records, or structures the name represents. However, these files and records can still be directly accessed. If you are using logical record locks, you must not simultaneously use other locking techniques. The logical record name is also referred to as a synchronization string or synch name.

You can specify a timeout value with locking requests. If any of the files or records are not available, the timeout value instructs the file server to wait for the files or records to become available. If the files or records are still not available at the end of the time specified by the timeout value, the locking request fails. This process reduces your number of retries and network traffic.

The timeout value is counted in units of approximately 1/18 of a second.

## Semaphores

NetWare provides system request that enable you to create, open, examine, and close semaphores. You can also use semaphore system requests to increment and decrement the value associated with a semaphore.

Like a logical record lock, a semaphore is a name associated with network resources such as files and records. Both logical record locks and semaphores limit the number of clients that can access network resources at one time. Logical record locks limit the number of clients to one.

Semaphores, however, allow a configurable number of clients (1 to 127) to access a network resource at one time.

When you create a semaphore using **Open Semaphore** 0x2222 32 0, you assign a value to the semaphore to indicate how many clients can simultaneously access the resource associated with the semaphore. For example, a semaphore value of 4 indicates that 5 (0 to 4) clients can access the associated resource at one time. All requests that access a semaphore must obtain a semaphore handle by first calling **Open Semaphore**.

After opening an existing semaphore, you must first check the semaphore's value by calling **Examine Semaphore** 0x2222 32 1. If the value is greater than or equal to zero, you can access the associated network resource. You then decrement the value by calling **Wait On Semaphore** 0x2222 32 2 so you can access the resource. After accessing the resource, you increment the semaphore value by calling **Signal Semaphore** 0x2222 32 3 and then exiting the semaphore.

If the semaphore value is negative, you cannot access the resource immediately. You can either wait a specified timeout interval for the resource to become available or you can retry later.

The following algorithm illustrates semaphore usage:

1. Open Semaphore
2. Examine Semaphore  
If semaphore value  $< 0$ , go to step 6.
3. Wait On Semaphore
4. Access the associated resource
5. Signal Semaphore
6. Close Semaphore

# 45 NCPs

This section describes each of the Synchronization NCPs, their Request and Reply formats, and Return Values.

# Clear File 0x2222 87 38

Releases the locks on a file and closes the file (if open).

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (38)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	reserved2 (0)	word
12	NWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Clear File** replaces **Clear File 0x2222 07** and uses NWHandlePathStruct to close a file instead of using only a 1-byte directory handle and a file name.

# Clear File (old) 0x2222 07

Clears one file from your logged file table.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (7)	byte
7	DirectoryHandle	byte
8	FileNameLen	byte
9	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
152	0x98	Disk Map Error
155	0x9B	Bad Directory Handle
156	0x9C	Invalid Path
161	0xA1	Directory I/O Errro
253	0xFD	Bad Station Number
255	0xFF	Unlock Error

## Remarks

If the specified file is open, it is closed and its file handle is invalidated. The file can then be accessed by other clients. If you opened the file multiple times, it is closed as many times and all associated file handles are invalidated.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set 0x2222 106 (page 1178), Clear File Set 0x2222 08 (page 1164), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201)**

# Clear File Set 0x2222 08

Clears all files from the client's logged file table.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (8)	byte
7	LockFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

All open files are closed and all the client's file handles are invalidated. File I/O cannot be performed while files are closed.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set 0x2222 106 (page 1178), Clear File (old) 0x2222 07 (page 1163), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201)**



# Clear Lock Wait Node 0x2222 112

Aborts a pending asynchronous lock by clearing it.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (112)	byte
7	WaitNode	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
255	0xFF	Lock Error

## Remarks

WaitNode points to [AllAttrStruc \(page 571\)](#) that has been allocated by calling on the following asynchronous locking NCPs:

- ♦ **Log File 0x2222 105**
- ♦ **Lock File Set 0x2222 106**
- ♦ **Log Logical Record 0x2222 107**
- ♦ **Lock Record Set 0x2222 108**
- ♦ **Log Physical Record 0x2222 109**
- ♦ **Log Physical Record Set 0x2222 110**

## See Also

[Log File \(old\) 0x2222 105 \(page 1185\)](#), [Lock File Set 0x2222 106 \(page 1178\)](#), [Log Logical Record 0x2222 107 \(page 1187\)](#), [Lock Logical Record Set 0x2222 108 \(page 1180\)](#), [Log Physical Record 0x2222 109 \(page 1191\)](#), [Lock Physical Record Set 0x2222 110 \(page 1182\)](#)

# Clear Logical Record 0x2222 11

Releases the specified synchronization string and removes it from your synchronization string table.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (11)	byte
7	SynchNameLen	byte
8	SynchName	byte[SynchNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Unlock Error

## Remarks

The synchronization string cannot be used until you relog it.

## See Also

**Clear Logical Record Set 0x2222 14 (page 1167), Log Logical Record 0x2222 107 (page 1187), Lock Logical Record Set 0x2222 108 (page 1180), Release Logical Record 0x2222 12 (page 1202), Release Logical Record Set 0x2222 13 (page 1203)**

# Clear Logical Record Set 0x2222 14

Releases all of your synchronization strings (logical record locks) and clear your synchronization string table.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (14)	byte
7	LockFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## See Also

[Clear Logical Record 0x2222 11 \(page 1166\)](#), [Log Logical Record 0x2222 107 \(page 1187\)](#), [Lock Logical Record Set \(old\) 0x2222 10 \(page 1181\)](#), [Release Logical Record 0x2222 12 \(page 1202\)](#), [Release Logical Record Set 0x2222 13 \(page 1203\)](#)

# Clear Physical Record 0x2222 87 69

Removes the specified byte range from the client's data byte range table.

**NetWare Servers:** 6.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (69)	byte
8	FileHandle	long (Hi-Lo)
12	LockAreaStartOffset	UINT64 (Hi-Lo)
20	LockAreaLength	UINT64 (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Bad Station Number
255	0xFF	Unlock Error

## Remarks

Note that *LockAreaStartOffset* and *LockAreaLength* in the request can be 64 bits. Requests made to NSS volumes support locking at true 64-bit offsets. Requests made to traditional volumes support locking at 32-bit offsets. Locking at offsets greater than 32 bits on a traditional volume result in an error.

*FileHandle*, *LockAreaStartOffset*, and *LockAreaLength* require input to be in Hi-Lo order.

If the byte range is locked, it is cleared. You cannot use the specified byte range until that range is again logged and locked.

The specified byte range must match a previously logged byte range. You cannot clear only a portion of a previously logged byte range.

## See Also

[Clear Physical Record Set 0x2222 31 \(page 1171\)](#),  
[Lock Physical Record Set 0x2222 110 \(page 1182\)](#),  
[Log Physical Record 0x2222 87 67 \(page 1189\)](#)

**Log Physical Record 0x2222 109 (page 1191),**  
**Release Physical Record 0x2222 87 68 (page 1204)**  
**Release Physical Record 0x2222 28 (page 1206),**  
**Release Physical Record Set 0x2222 29 (page 1207)**

# Clear Physical Record 0x2222 30

Removes the specified byte range from the client's data byte range table.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (30)	byte
7	Reserved	byte
8	FileHandle	byte[6]
14	LockAreaStartOffset	long (Hi-Lo)
18	LockAreaLen	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Unlock Error

## Remarks

If the byte range is locked, it is cleared. You cannot use the specified byte range until that range is again logged and locked.

The specified byte range must match a previously logged byte range. You cannot clear only a portion of a previously logged byte range.

## See Also

**Clear Physical Record Set 0x2222 31 (page 1171), Log Physical Record 0x2222 109 (page 1191), Lock Physical Record Set 0x2222 110 (page 1182), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207)**

# Clear Physical Record Set 0x2222 31

Clears the client's data byte range table of all byte ranges held by the file server for the calling task.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (31)	byte
7	LockFlags	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

All locked ranges are released.

## See Also

**Clear Physical Record 0x2222 30 (page 1170), Log Physical Record 0x2222 109 (page 1191), Lock Physical Record Set 0x2222 110 (page 1182), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207)**

# Close Semaphore 0x2222 111 04

Closes a semaphore that is no longer needed.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (111)	byte
7	SubFunctionCode (01)	byte
8	SemaphoreHandle	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	OpenCount	byte
9	ShareCount	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## Remarks

The corresponding semaphore *OpenCount* (the number of clients accessing the semaphore) is decremented by one. If the semaphore *OpenCount* is zero after **Close Semaphore** completes, the server deletes the semaphore.

## See Also

**Open/Create a Semaphore 0x2222 111 00 (page 1197), Examine Semaphore 0x2222 111 01 (page 1174)**



# Close Semaphore (old) 0x2222 32 04

Closes a semaphore that is no longer needed.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (32)	byte
7	SubFunctionCode (4)	byte
8	SemaphoreHandle	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## Remarks

The corresponding semaphore *OpenCount* (the number of clients accessing the semaphore) is decremented by one. If the semaphore *OpenCount* is zero after **Close Semaphore** completes, the server deletes the semaphore.

## See Also

**Examine Semaphore 0x2222 111 01 (page 1174)**

# Examine Semaphore 0x2222 111 01

Returns the current value of the target semaphore.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (111)	byte
7	SubFunctionCode (01)	byte
8	SemaphoreHandle	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	SemaphoreValue	byte
9	SemaphoreOpenCount	byte

## Parameters

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore.

*SemaphoreOpenCount*

(Reply) Specifies the number of clients that are using the semaphore.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## See Also

[Close Semaphore 0x2222 111 04 \(page 1172\)](#)

# Examine Semaphore (old) 0x2222 32 01

Returns the current value of the target semaphore.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (32)	byte
7	SubFunctionCode (1)	byte
8	SemaphoreHandle	long (Lo-Hi)

## Reply Format

Offset	Content	Type
Reply header		
8	SemaphoreValue	byte
9	SemaphoreOpenCount	byte

## Parameters

*SemaphoreValue*

(Reply) Specifies the current value of the semaphore.

*SemaphoreOpenCount*

(Reply) Specifies the number of clients that are using the semaphore.

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## See Also

**Close Semaphore (old) 0x2222 32 04 (page 1173)**

# File Set Lock (old) 0x2222 01

Locks shareable files that the calling station has open.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (1)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**File Set Lock (old)** is provided for compatibility with previous NetWare versions.

## See Also

**File Release Lock 0x2222 02 (page 1177)**

# File Release Lock 0x2222 02

Releases locks on shareable files that were locked by the last **File Set Lock** request.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (2)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**File Release Lock** is provided for compatibility with previous NetWare versions.

## See Also

**File Set Lock (old) 0x2222 01 (page 1176)**

# Lock File Set 0x2222 106

Locks all files logged by your current task.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (106)	byte
7	LockTimeout	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
255	0xFF	Lock Error

## Remarks

If all files cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*. If all files cannot be locked, no files are locked.

## See Also

**Log File (old) 0x2222 105 (page 1185), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164)**

## Lock File Set (old) 0x2222 04

Locks all files logged by your current task.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (4)	byte
7	LockTimeout	word (Lo-Hi)

### Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	Timeout
255	0xFF	Lock Error

### Remarks

If all files cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*. If all files cannot be locked, no files are locked.

### See Also

**Log File (old) 0x2222 105 (page 1185), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164), Lock File Set (old) 0x2222 04 (page 1179)**

# Lock Logical Record Set 0x2222 108

Locks all your logged synchronization strings.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (108)	byte
7	LockFlag	byte
8	LockTimeout	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
255	0xFF	Lock Error

## Remarks

If all files cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*. If all files cannot be locked, **Lock Logical Record Set** fails and returns Lock Error. If all files cannot be locked, none of the records in the set are locked.

If bit 1 of *LockFlag* is set, the lock types are exclusive (read-write); otherwise, they are shareable (read only).

## See Also

**Log Logical Record 0x2222 107 (page 1187), Release Logical Record 0x2222 12 (page 1202), Release Logical Record Set 0x2222 13 (page 1203), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167), Lock Logical Record Set 0x2222 108 (page 1180)**



# Lock Logical Record Set (old) 0x2222 10

Locks all your logged synchronization strings.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (10)	byte
7	LockFlag	byte
8	LockTimeout	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

If all files cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*. If all files cannot be locked, **Lock Logical Record Set (old)** fails and returns Lock Error. If all files cannot be locked, none of the records in the set are locked.

## See Also

**Log Logical Record (old) 0x2222 09 (page 1188), Release Logical Record 0x2222 12 (page 1202), Release Logical Record Set 0x2222 13 (page 1203), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167), Lock Logical Record Set 0x2222 108 (page 1180)**

# Lock Physical Record Set 0x2222 110

Locks all byte ranges that you currently have logged.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (110)	byte
7	LockFlag	byte
8	LockTimeout	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	Waiting
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

If all byte ranges cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*.

If the lock is currently available, the server returns Waiting, which frees the requester so it can send other NCP requests for other session.

When you timeout or the lock becomes available, the server sends an out-of-bounds packet (3rd socket) to the requester. The requester then sends 0x2222 112 to find out the status.

If bit 1 of *LockFlag* is set, the lock types are exclusive (read-write); otherwise, they are shareable (read only).

## See Also

**Log Physical Record 0x2222 109 (page 1191), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171), Lock Physical Record Set 0x2222 110 (page 1182)**

# Lock Physical Record Set (old) 0x2222 27

Locks all byte ranges that you currently have logged.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (27)	byte
7	LockFlag	byte
8	LockTimeout	word (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

If all byte ranges cannot be locked immediately, the file server will retry the request for the amount of time specified in *LockTimeout*.

If bit 1 of *LockFlag* is set, the lock types are exclusive (read-write); otherwise, they are shareable (read only).

## See Also

**Log Physical Record (old) 0x2222 26 (page 1193), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171), Lock Physical Record Set 0x2222 110 (page 1182)**

# Log File 0x2222 87 36

Accesses a file using the NWHandlePathStruct, rather than using only a 1-byte directory handle and a file name.

**NetWare Servers:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (36)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	reserved2 (0)	word (Lo-Hi)
12	LogFileFlags	long (Lo-Hi)
16	WaitTime	long (Lo-Hi)
20	NWHandlePathStruct	structure

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

**Log File** replaces **Log File 0x2222 03** and **Log File 0x2222 105**.

The callback capabilities provided by **Log File 0x2222 105** are accessible by enabling the callback in *LogFileFlag*.

*LogFileFlag* can have the following values:

0x00000001 LockFileImmediatelyBit

0x00008000 CallBackRequestedBit

# Log File (old) 0x2222 105

Logs the specified file for your exclusive use.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (105)	byte
7	DirectoryHandle	byte
8	LockFlag	byte
9	LockTimeout	word (Hi-Lo)
11	FileNameLen	byte
12	FileName	byte[255]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
130	0x82	No Open Privileges
150	0x96	Server Out Of Memory
255	0xFF	Lock Error

## Remarks

If bit 0 of *LockFlag* is set, the server will immediately attempt to lock the file. You can lock a file even if the file does not yet exist; this reserves the file name for you.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set 0x2222 106 (page 1178), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164)**

# Log File (old) 0x2222 03

Logs the specified file for your exclusive use.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (3)	byte
7	DirectoryHandle	byte
8	LockFlag	byte
9	LockTimeout	word (Hi-Lo)
11	FileNameLen	byte
12	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
130	0x82	No Open Privileges
150	0x96	Server Out Of Memory
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

If bit 0 of *LockFlag* is set, the server will immediately attempt to lock the file. You can lock a file even if the file does not yet exist; this reserves the file name for you.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set (old) 0x2222 04 (page 1179), Release File (old) 0x2222 05 (page 1200), Release File Set 0x2222 06 (page 1201), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164)**

# Log Logical Record 0x2222 107

Logs a synchronization string for your use.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (107)	byte
7	LockFlag	byte
8	LockTimeout	word (Hi-Lo)
10	SynchNameLen	byte
11	SynchName	byte[255]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	Lock Waiting Error
150	0x96	Server Out Of Memory
255	0xFF	Lock Error

## Remarks

The string can be up to 128 characters long.

If bit 0 of *LockFlag* is set, the server will immediately attempt to lock the file. *LockTimeout* determines how long the server will try to lock the string before reporting failures.

## See Also

**Log Logical Record 0x2222 107 (page 1187), Lock Logical Record Set (old) 0x2222 10 (page 1181)L, Release Logical Record 0x2222 12 (page 1202), Release Logical Record Set 0x2222 13 (page 1203), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167)**

# Log Logical Record (old) 0x2222 09

Logs a synchronization string for your use.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (9)	byte
7	LockFlag	byte
8	LockTimeout	word (Hi-Lo)
10	SynchNameLen	byte
11	SynchName	byte[SynchNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

The string can be up to 128 characters long.

If bit 0 of *LockFlag* is set, the server will immediately attempt to lock the file. *LockTimeout* determines how long the server will try to lock the string before reporting failures.

## See Also

**Log Logical Record 0x2222 107 (page 1187), Lock Logical Record Set (old) 0x2222 10 (page 1181), Release Logical Record Set 0x2222 13 (page 1203), Release Logical Record 0x2222 12 (page 1202), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167)**



# Log Physical Record 0x2222 87 67

Locks a byte range within a file for your use.

**NetWare Servers:** 6.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (67)	byte
8	LockFlag	long (Lo-Hi)
12	FileHandle	long (Hi-Lo)
16	LockAreasStartOffset	UINT64 (Hi-Lo)
24	LockAreaLen	UINT64 (Hi-Lo)
32	LockTimeout	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
136	0x88	Invalid File Handle
150	0x96	Server Out Of Memory
253	0xFD	Lock Collision Bad Station Number
255	0xFF	Lock Error

## Remarks

Note that *LockAreaStartOffset* and *LockAreaLength* in the request can be 64 bits. Requests made to NSS volumes support locking at true 64-bit offsets. Requests made to traditional volumes support locking at 32-bit offsets. Locking at offsets greater than 32 bits on a traditional volume result in an error.

*FileHandle*, *LockAreaStartOffset*, *LockAreaLength*, and *LockTimeout* require input to be in Hi-Lo order. *LockFlag* requires input to be in Lo-Hi order.

*LockFlag* can have the following values:

- 00h Byte range is not locked, but logged for future use
- 01h Byte range is locked with exclusive (read-write) access
- 03h Byte range is locked with shareable (read-only) access

*LockTimeout* specifies how long the file server will attempt to lock the byte range before returning failures. *LockTimeout* is valid only if *LockFlag* is 01h or 03h.

Locked byte ranges can be used only by the task making the lock request. Overlapping lock ranges are allowed.

## See Also

[Clear Physical Record 0x2222 87 69 \(page 1168\)](#),  
[Clear Physical Record 0x2222 30 \(page 1170\)](#),  
[Clear Physical Record Set 0x2222 31 \(page 1171\)](#),  
[Lock Physical Record Set 0x2222 110 \(page 1182\)](#),  
[Log Physical Record 0x2222 109 \(page 1191\)](#),  
[Release Physical Record 0x2222 87 68 \(page 1204\)](#),  
[Release Physical Record Set 0x2222 29 \(page 1207\)](#)

# Log Physical Record 0x2222 109

Locks a byte range within a file for your use.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (109)	byte
7	LockFlag	byte
8	NWFileHandle	byte[6]
14	LockAreasStartOffset	long (Hi-Lo)
18	LockAreaLen	long (Hi-Lo)
22	LockTimeout	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
127	0x7F	ERR_LOCK_WAITING
136	0x88	Invalid File Handle
150	0x96	Server Out Of Memory
253	0xFD	Lock Collision
255	0xFF	Lock Error

## Remarks

*LockFlag* can have the following values:

00h Byte range is not locked, but logged for future use

01h Byte range is locked with exclusive (read-write) access

03h Byte range is locked with shareable (read-only) access

*LockTimeout* specifies how long the file server will attempt to lock the byte range before returning failures. *LockTimeout* is valid only if *LockFlag* is 01h or 03h.

Locked byte ranges can be used only by the task making the lock request. Overlapping lock ranges are allowed.

## See Also

**Log Physical Record 0x2222 109 (page 1191), Lock Physical Record Set 0x2222 110 (page 1182), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171)**

# Log Physical Record (old) 0x2222 26

Locks a byte range within a file for your use.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (26)	byte
7	LockFlag	byte
8	FileHandle	byte[6]
14	LockAreasStartOffset	long (Hi-Lo)
18	LockAreaLen	long (Hi-Lo)
22	LockTimeout	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
136	0x88	Invalid File Handle
150	0x96	Server Out Of Memory
253	0xFD	Lock Collision
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

*LockFlag* can have the following values:

00h Byte range is not locked, but logged for future use

01h Byte range is locked with exclusive (read-write) access

03h Byte range is locked with shareable (read-only) access

*LockTimeout* specifies how long the file server will attempt to lock the byte range before returning failures. *LockTimeout* is valid only if *LockFlag* is 01h or 03h.

Locked byte ranges can be used only by the task making the lock request. Overlapping lock ranges are allowed.

## See Also

**Lock Physical Record Set (old) 0x2222 27 (page 1183), Release Physical Record 0x2222 28 (page 1206), Release Physical Record Set 0x2222 29 (page 1207), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171)**

# Open Semaphore (old) 0x2222 32 00

Opens a named semaphore.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (32)	byte
7	SubFunctionCode (0)	byte
8	InitialSemaphoreValue	byte
9	SemaphoreNameLen	byte
10	SemaphoreName	byte[SemaphoreNameLen]

## Reply Format

Offset	Content	Type
Reply header		
8	SemaphoreHandle	long Lo-Hi)
9	SemaphoreOpenCount	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
255	0xFF	Lock Error

## Remarks

If the semaphore does not exist, the server creates one. If the semaphore already exists, the *InitialSemaphoreValue* is ignored.

You must use the returned *SemaphoreHandle* when accessing the semaphore.

*SemaphoreOpenCount* specifies how many clients currently have the semaphore open. If you created this semaphore, *SemaphoreOpenCount* is 1.

## See Also

**Close Semaphore (old) 0x2222 32 04 (page 1173), Open/Create a Semaphore 0x2222 111 00 (page 1197)**



# Open/Create a Semaphore 0x2222 111 00

Opens a named semaphore.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (111)	byte
7	SubFunctionCode (0)	byte
8	InitialSemaphoreValue	byte
9	SemaphoreNameLen	byte
10	SemaphoreName	byte[512]

## Reply Format

Offset	Content	Type
Reply header		
8	SemaphoreHandle	long (Lo-Hi)
12	SemaphoreOpenCount	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory
255	0xFF	Lock Error

## Remarks

If the semaphore does not exist, the server creates one. If the semaphore already exists, the *InitialSemaphoreValue* is ignored.

You must use the returned *SemaphoreHandle* when accessing the semaphore.

*SemaphoreOpenCount* specifies how many clients currently have the semaphore open. If you created this semaphore, *SemaphoreOpenCount* is 1.

## See Also

[Close Semaphore \(old\) 0x2222 32 04 \(page 1173\)](#)

## Release File 0x2222 87 37

Releases the locks on a file using NWHandlePathStruct, rather than using only a 1-byte directory handle and a file name.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (37)	byte
8	NameSpace	byte
9	reserved (0)	byte
10	reserved2 (0)	word
12	NWHandlePathStruct	structure

### Return Values

Decimal	Hex	Description
0	0x00	Successful

### Remarks

**Release File** replaces **Release File 0x2222 05**.

# Release File (old) 0x2222 05

Releases the specified file that you previously locked.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (5)	byte
7	DirectoryHandle	byte
8	FileNameLen	byte
9	FileName	byte[FileNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
156	0x9C	Invalid Path
255	0xFF	Unlock Error

## Remarks

Releasing a file allows it to be used by other clients.

If the file is open, the file handle becomes invalid.

The file remains in your logged file table and be locked by future requests to **Lock File Set 0x2222 04**. You cannot perform any file I/O on the file until you lock it again.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set 0x2222 106 (page 1178), Release File Set 0x2222 06 (page 1201), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164)**

# Release File Set 0x2222 06

Releases all of your currently logged or locked files for use by other clients.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (6)	byte
7	LockFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

File handle for any open files become invalid.

You will not be able to perform any file I/O to the released files until they are locked again by calling **Lock File Set 0x2222 04**. The files will remain in your logged file table.

## See Also

**Log File (old) 0x2222 105 (page 1185), Lock File Set 0x2222 106 (page 1178), Release File (old) 0x2222 05 (page 1200), Clear File (old) 0x2222 07 (page 1163), Clear File Set 0x2222 08 (page 1164)**

# Release Logical Record 0x2222 12

Releases one of your synchronization strings.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (12)	byte
7	SynchNameLen	byte
8	SynchName	byte[SynchNameLen]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Unlock Error

## Remarks

Releasing logical records allows other clients to lock the records. The string remains in your synchronization string table and will be relocked when you call **Lock Logical Record Set 0x2222 10**.

## See Also

**Log Logical Record 0x2222 107 (page 1187), Lock Logical Record Set (old) 0x2222 10 (page 1181), Release Logical Record Set 0x2222 13 (page 1203), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167)**

# Release Logical Record Set 0x2222 13

Releases all of your locked synchronization strings for use by other clients.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (13)	byte
7	LockFlag	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The released strings remain in your synchronization string table and will be relocked when you call **Lock Logical Record Set 0x2222 10**.

## See Also

**Log Logical Record 0x2222 107 (page 1187), Lock Logical Record Set (old) 0x2222 10 (page 1181), Clear Logical Record 0x2222 11 (page 1166), Clear Logical Record Set 0x2222 14 (page 1167), Release Logical Record 0x2222 12 (page 1202)**

# Release Physical Record 0x2222 87 68

Releases a byte range that you previously locked.

**NetWare Servers:** 6.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (87)	byte
7	SubFunctionCode (68)	byte
8	FileHandle	long (Hi-Lo)
12	LockAreaStartOffset	UINT64 (Hi-Lo)
20	LockAreaLen	UINT64 (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Bad Station Number
255	0xFF	Unlock Error

## Remarks

Note that *LockAreaStartOffset* and *LockAreaLength* in the request can be 64 bits. Requests made to NSS volumes support locking at true 64-bit offsets. Requests made to traditional volumes support locking at 32-bit offsets. Locking at offsets greater than 32 bits on a traditional volume result in an error.

*FileHandle*, *LockAreaStartOffset*, *LockAreaLength* require input to be in Hi-Lo order.

The released byte range remains in your logged data block table and will be relocked when you call **Lock Physical Record Set 0x2222 27**.

Release byte ranges must match locked byte ranges. Releasing only a portion of a previously locked byte range is not allowed. If the byte range being released overlaps any other byte range locks still in effect, all but the overlapped bytes will be released.

## See Also

[Clear Physical Record 0x2222 87 69 \(page 1168\)](#)

[Clear Physical Record 0x2222 30 \(page 1170\)](#)



**Clear Physical Record Set 0x2222 31 (page 1171),**  
**Lock Physical Record Set 0x2222 110 (page 1182),**  
Log Physical Record 0x2222 87 67 (page 1189)  
**Log Physical Record 0x2222 109 (page 1191),**  
**Release Physical Record Set 0x2222 29 (page 1207)**

# Release Physical Record 0x2222 28

Releases a byte range that you previously locked.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (28)	byte
7	Reserved	byte
8	FileHandle	byte[6]
14	LockAreaStartOffset	long (Hi-Lo)
18	LockAreaLen	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Unlock Error

## Remarks

The released byte range remains in your logged data block table and will be relocked when you call **Lock Physical Record Set 0x2222 27**.

Release byte ranges must match locked byte ranges. Releasing only a portion of a previously locked byte range is not allowed. If the byte range being released overlaps any other byte range locks still in effect, all but the overlapped bytes will be released.

## See Also

**Log Physical Record 0x2222 109 (page 1191), Lock Physical Record Set 0x2222 110 (page 1182), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171), Release Physical Record Set 0x2222 29 (page 1207)**

# Release Physical Record Set 0x2222 29

Releases all byte ranges that you have previously locked.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (29)	byte
7	LockFlags	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

The released byte ranges remain in your logged byte range table and will be relocked when you call **Lock Physical Record Set 0x2222 27**.

## See Also

**Log Physical Record 0x2222 109 (page 1191), Lock Physical Record Set 0x2222 110 (page 1182), Clear Physical Record 0x2222 30 (page 1170), Clear Physical Record Set 0x2222 31 (page 1171), Release Physical Record 0x2222 28 (page 1206)**

# Signal Semaphore (old) 0x2222 32 03

Increments the semaphore value by one.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (32)	byte
7	SubFunctionCode (3)	byte
8	SemaphoreHandle	long (Lo-Hi)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## Remarks

If another client is waiting on the semaphore, Successful is returned to the waiting client.

## See Also

**Wait On (P) Semaphore 0x2222 111 02 (page 1211), Signal (V) Semaphore 0x2222 111 03 (page 1209)**

## Signal (V) Semaphore 0x2222 111 03

Increments the semaphore value by one.

**NetWare Servers:** 3.x, 4.x, 5.x

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (111)	byte
7	SubFunctionCode (03)	byte
8	SemaphoreHandle	long (Lo-Hi)

### Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

### Remarks

If another client is waiting on the semaphore, Successful is returned to the waiting client.

### See Also

**Wait On (P) Semaphore 0x2222 111 02 (page 1211)**

# Wait On Semaphore (old) 0x2222 32 02

Allows a client to wait for a semaphore.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (32)	byte
7	SubFunctionCode (02)	byte
8	SemaphoreHandle	long (Lo-Hi)
12	SemaphoreTimeOut	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

The semaphore value is decremented by one. If the resulting value is greater than or equal to zero, Successful is returned. If the resulting value is less than zero, the server waits for a **Signal Semaphore** 0x2222 32 03 request for the amount of time specified by *SemaphoreTimeOut*.

If no signal arrives, the server automatically reincrements the semaphore value and returns Timeout, indicating that the semaphore is not available.

## See Also

**Signal Semaphore (old) 0x2222 32 03 (page 1208), Wait On (P) Semaphore 0x2222 111 02 (page 1211)**

# Wait On (P) Semaphore 0x2222 111 02

Allows a client to wait for a semaphore.

**NetWare Servers:** 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (111)	byte
7	SubFunctionCode (02)	byte
8	SemaphoreHandle	long (Lo-Hi)
12	SemaphoreTimeOut	word (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
254	0xFE	Timeout
255	0xFF	Lock Error

## Remarks

The semaphore value is decremented by one. If the resulting value is greater than or equal to zero, Successful is returned. If the resulting value is less than zero, the server waits for a **Signal Semaphore** 0x2222 32 03 request for the amount of time specified by *SemaphoreTimeOut*.

If no signal arrives, the server automatically reincrements the semaphore value and returns Timeout, indicating that the semaphore is not available.

## See Also

**Signal (V) Semaphore 0x2222 111 03 (page 1209)**





# 46 Structures

This section describes the Synchronization structures and their fields.

# ASyncLockWaitStruct

Contains information about asynchronous locking.

## Syntax

```
struct ASyncLockWaitStructure
{
    struct ASyncLockWaitStructure  *ALink;
    LONG                           AStation;
    LONG                           ATask;
    LONG                           ACompletionCode;
    LONG                           ALastTime;
    LONG                           ALastTime;
    BYTE                           AStatus;
    BYTE                           AWaitType;
};
```

## Fields

### *ALink*

Points to the next ASyncLockWaitStruct in a linked list.

### *AStation*

Specifies the connection that allocated a structure.

### *ATask*

Specifies the task number for which a structure has been allocated.

### *ACompletionCode*

Specifies the completion code.

### *ALastTime*

Specifies the wake up time for the *WaitNode*.

### *AStatus*

Specifies the status of the *WaitNode*:

- 0 Inserting
- 1 Waiting
- 2 Completed

### *AWaitType*

Specifies the type of WaitNode lock:

- 0 Physical Record Lock
- 1 NonPhysical Record Lock

# XX Time Synchronization

To access Time Synchronization NCPs, use the following:

- ♦ [Chapter 47, “NCPs,” on page 1217](#)
- ♦ [Chapter 48, “Structures,” on page 1225](#)



# 47

## NCPs

This section describes each of the Time Synchronization NCPs, their Request and Reply formats, and Return Values.

# Timesync Exchange Time 0x2222 114 02

Returns the UTC time (or the number of seconds since January 1, 1970).

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (114)	byte
7	SubFuncStrucLen (103)	word (Hi-Lo)
9	SubFuncCode (02)	byte
10	protocolFlags	long
14	nodeFlags	long
18	sourceOriginateTime	int64
26	targetReceiveTime	int64
34	targetTransmitTime	int64
42	sourceReturnTime	int64
50	eventOffset	int64
58	eventTime	long
62	serverName	char[50]

## Reply Format

Offset	Content	Type
Reply header		
8	SubFuncStrucLen (103)	word (Hi-Lo)
10	SubFuncCode (02)	byte
11	protocol Flags	long
15	nodeFlags	long
19	sourceOriginateTime	int64
27	targetReceiveTime	int64
35	targetTransmitTime	int64

Offset	Content	Type
43	sourceReturnTime	int64
51	eventOffset	int64
59	eventTime	long
63	serverName	char[50]

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	NCP Boundary Check Failed
251	0xFB	Invalid Subfunction Request

## Remarks

**TimeSync Exchange Time** can be called by other servers (primary, secondary, etc.) and by clients to return the time. Since the client that originates the call knows how the data can be used, the client can choose to ignore fields that do not apply.

**IMPORTANT:** The server name must be NULL-terminated, which is why the structure has an extra 49th byte. If the server name is not NULL terminated, this server attempts to notify that server when a network event is received (reinitialize, set time, etc.).

Note that [ObjectInfoStruc \(page 174\)](#) is used in both the request and reply fields. This structure is modified to reflect the actual target data.

# Timesync Get Server List 0x2222 114 05

Returns the server list.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (114)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (05)	byte
10	StartNumber	long

## Reply Format

Offset	Content	Type
Reply header		
8	nameType	word
20	serverName	char[48]
28	serverListFlags	long
32	startNumberFlag	word

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	NCP Boundary Check Failed
251	0xFB	Invalid Subfunction Request

## See Also

[Timesync Set Server List 0x2222 114 06 \(page 1224\)](#)



# Timesync Get Time 0x2222 114 01

Returns the UTC time (or the number of seconds since January 1, 1970) from the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (114)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (01)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	theTime	long[3]
20	eventOffset	int64
28	eventTime	long
32	eventParm	void *

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	NCP Boundary Check Failed
251	0xFB	Invalid Subfunction Request

## Remarks

You can call this NCP to return the current clock, status, and information about any scheduled network event and then use the returned information to update a workstation's time.

This NCP replaces NCP 20 functionality, which is not used by current Novell clients.

*theTime* is made up of three longs (which should each be long swapped): 1) is the UTC time in seconds; 2) is the fractional seconds; 3) is the timesync status flags:

0x00000001	CLOCK_IS_SYNCHRONIZED
0x00000002	CLOCK_IS_NETWORK_SYNCHRONIZED
0x00000004	CLOCK_SYNCRHONIZATION_IS_ACTIVE
0x00000008	CLOCK_EXTERNAL_SYNC_ACTIVE
0x00000000	SERVER_TYPE_UNKNOWN
0x00000100	SERVER_TYPE_CLIENT
0x00000200	SERVER_TYPE_SECONDARY
0x00000300	SERVER_TYPE_PRIMARY
0x00000400	SERVER_TYPE_REFERENCE
0x00000500	SERVER_TYPE_SINGLE_SERVER
0x000F0000	CLOCK_STATUS_EXTRN_SYNC_TYPE

Note that the *eventOffset* reply field is defined as a 64-bit field.

# Timesync Get Version 0x2222 114 12

Returns the time synchronization version from the server.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (114)	byte
7	SubFuncStrucLen (1)	word (Hi-Lo)
9	SubFuncCode (12)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	version	long

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	NCP Boundary Check Failed
251	0xFB	Invalid Subfunction Request

# Timesync Set Server List 0x2222 114 06

Sets the server list.

**NetWare Server:** 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (114)	byte
7	SubFuncStrucLen (5)	word (Hi-Lo)
9	SubFuncCode (06)	byte
10	StartNumber	long

## Reply Format

Offset	Content	Type
Reply header		
8	nameType	word
20	serverName	char[48]
28	serverListFlags	long
32	startNumberFlag	word

## Return Values

Decimal	Hex	Description
0	0x00	Successful
126	0x7E	NCP Boundary Check Failed
251	0xFB	Invalid Subfunction Request

## Remarks

**WARNING:** This NCP returns zero or success in all cases, regardless of input.

## See Also

[Timesync Get Server List 0x2222 114 05 \(page 1220\)](#)

# 48 Structures

This section describes the Time Synchronization structures and their fields.

# Exchange\_Time\_Struct

Returns the specified time information.

## Syntax

```
struct Exchange_Time_Struct
{
    /* Subfunction information is to complete NCP header */
    WORD          SubFunctionLength;
    BYTE          SubFunctionCode;
    LONG          protocolFlags; /* of source */
    LONG          nodeFlags;     /* of source & target */
    int64         sourceOriginateTime;
    int64         targetReceiveTime;
    int64         targetTransmitTime;
    int64         sourceReturnTime;
    /* seconds only - sub seconds used for flags */
    int64         eventOffset;
    LONG          eventTime; /* seconds only */
    /* source on input, target on output; size = 50 bytes */
    PaddedServerName  serverName;
} Exchange_Time_Struct;
/* total size of structure = 105 bytes */
```



TTS NCPs allow NetWare file servers to track transactions and ensure file integrity by backing out (or erasing) interrupted or partially completed transactions. TTS tracks transactions only for transactional files. A file becomes transactional when the file's Transactional extended file attribute is set. (See **Set File Extended Attribute** 0x2222 79 and **Scan File Information** 0x2222 23 15.)

For example, a banking database application frequently performs a transaction that includes three writes to database files: a debit to one account, a credit to another account, and a note to a log. The application must complete all three writes to maintain database integrity. However, a hardware problem or power failure can interrupt such a transaction and cause database corruption. TTS prevents database corruption in such events by allowing the transaction to be erased.

To access TTS NCPs, use the following:

- ♦ [Chapter 49, “Concepts,” on page 1229](#)
- ♦ [Chapter 50, “NCPs,” on page 1231](#)





# 49 Concepts

This section explains ideas that are common to TTS NCPs.

## Implicit vs. Explicit Tracking

TTS is divided into two categories: implicit transaction tracking and explicit transaction tracking.

Implicit transaction tracking requires no special coding on your part. If TTS is installed and enabled on a file server, TTS tracks all transactions to all transactional files (including transactions made by NetWare to bindery files).

Explicit transaction tracking requires applications to make TTS system requests. By calling both TTS and locking system requests, an application can bracket file update sequences. An application will most likely use logical or physical record locks in conjunction with TTS system calls.

## Transaction Process

The following steps describe how TTS tracks each write within a transaction:

1. An application writes new data to a file on a file server.
2. The file server stores the new data in cache memory.

The target file on the file server hard disk remains unchanged.

3. The file server scans the target file on the file server hard disk, finds the data to be changed (old data), and copies the old data to cache memory.
4. The file server records the name and directory path of the target file and the location and length of the old data (record) within the file.

The target file on the file server hard disk is still unchanged.

5. The file server writes the old data in cache memory to a transaction work file on the file server hard disk.

The transaction work file resides at the root level of a volume on the file server (specified when generating the file server OS).

6. The transaction work file is flagged System and Hidden.

The target file on the file server hard disk is still unchanged.

7. The file server writes the new data in cache memory to the target file on the file server hard disk.

The target file is now changed.

The file server repeats these steps for each write within a transaction. The transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, the file server

writes the contents of the transaction work file to the target file and restores the file to its pretransaction condition. In effect, the file server backs out the transaction.

## Transaction Monitoring

The file server can monitor from 1 to 200 transactions at a time. This value is configurable when the file server OS is generated and can be set to 50 through 200.

A file server can track only one transaction at a time for each workstation DOS task. The file server places transactions that are not being serviced in a queue.

# 50 NCPs

This section describes each of the TTS NCPs, their Request and Reply formats, and Return Values.

# TTS Abort Transaction 0x2222 34 03

Aborts explicit and implicit transactions.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (3)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
253	0xFD	Transaction Tracking Disabled
254	0xFE	Transaction Restart
255	0xFF	Lock Error

## Remarks

After **TTS Begin Transaction** completes, TTS tracks all files that are currently open, as well as any files that are opened during the transaction. When an application writes to a transaction file, the file server automatically generates a physical record lock for any file that isn't already locked.

Locked files remain locked. Transaction files are not closed and unlocked until **TTS End Transaction** or **TTS Abort Transaction** executes.

If transaction files are updated, logical and physical record locks are held until the end of the transaction. If a transaction file is not updated, the logical or physical lock on that file can be released.

## See Also

**TTS End Transaction 0x2222 34 02 (page 1234)**

# TTS Begin Transaction 0x2222 34 01

Begins an explicit transaction.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (1)	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If a transaction is aborted, all writes made since the beginning of a transaction are cancelled, and all files are returned to the state they were in before the transaction began.

**TTS Abort Transaction** releases the following record locks:

- ♦ Physical record locks generated by the file server when an application tried to write to an unlocked record
- ♦ Physical or logical locks that have not been released because of a file write

For explicit transactions, the transaction is backed out; any locks being held are released.

For implicit transactions, if the number of logical or physical records still locked exceeds the threshold, **TTS Abort Transaction** returns Transaction Restart (0xFE), the transaction is backed out, and the file server automatically starts a new implicit transaction.

## See Also

**TTS End Transaction 0x2222 34 02 (page 1234)**

# TTS End Transaction 0x2222 34 02

Ends an explicit transaction and returns a transaction number.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (2)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	TransactionNumber	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful
255	0xFF	Lock Error

## Remarks

A transaction is not necessarily written to disk when **TTS End Transaction** returns. Pass the returned *TransactionNumber* to **TTS Transaction Status** (0x2222 34 4) to verify a successful transaction completion to disk.

If the file server fails before all transaction updates are written to disk, the transaction will be backed out when the file server is rebooted, unless transaction tracking is disabled.

**TTS End Transaction** releases the following record locks:

- ♦ Physical record locks generated by the file server when an application tried to write to an unlocked record
- ♦ Physical or logical locks that have not been released because of a file write

For explicit transactions, the transaction is backed out; any locks being held are released.

For implicit transactions, if the number of logical or physical records still locked exceeds the threshold, **TTS Abort Transaction** returns Transaction Restart (0xFE), the transaction is backed out, and the file server automatically starts a new implicit transaction.

## See Also

**TTS Begin Transaction 0x2222 34 01 (page 1233)**, **TTS Abort Transaction 0x2222 34 03 (page 1232)**, **TTS Transaction Status 0x2222 34 04 (page 1247)**

# TTS Get Application Thresholds 0x2222 34 05

Returns the number of logical and physical record locks allowed before an implicit transaction begins.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (5)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	LogicalLockThreshold	byte
9	PhysicalLockThreshold	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If a threshold value is zero, an implicit transaction will begin when the first lock is made for that particular lock type; if a threshold value is 100, an implicit transaction will begin when the one hundred and first lock is made, etc.

**TTS Get Application Thresholds** and **TTS Set Application Thresholds** (0x2222 34 6) can be used to change the implicit application thresholds and to restore the thresholds.

For example, **TTS Get Application Thresholds** can query an application for the number of logical and physical record locks allowed before an implicit transaction begins. **TTS Set Application Thresholds** can then do either of the following:

- ◆ Turn off implicit transactions
- ◆ Set implicit thresholds for applications that always keep one or more records locked



Applications that use only explicit transactions, but that sometimes generate unnecessary implicit transactions, should turn off all implicit transactions.

The default thresholds for logical and physical locks is zero. A threshold value of 0xFF means implicit transaction for that lock type is not in effect.

The returned thresholds are valid only for the requesting application. When the application terminates, the workstation thresholds are restored.

## See Also

**[TTS Set Application Thresholds 0x2222 34 06 \(page 1242\)](#)**

# TTS Get Transaction Bits 0x2222 34 09

Returns the transaction bits (*ControlFlags*) for a task.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (9)	byte

## Reply Format

Offset	Content	Type
Reply header		
9	ControlFlags	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Currently, only bit 0 is used. If bit 0 of the *ControlFlags* byte is set, forced record locking is enabled. The default state of forced record locking is enabled. Bits 1 through 6 are currently reserved.

## See Also

**TTS Set Transaction Bits 0x2222 34 10 (page 1244)**

# TTS Get Workstation Thresholds 0x2222 34 07

Returns the number of logical and physical record locks allowed for a workstation before an implicit transaction begins.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (7)	byte

## Reply Format

Offset	Content	Type
Reply header		
8	LogicalLockThreshold	byte
9	PhysicalLockThreshold	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If a threshold value is zero, an implicit transaction will begin when the first lock is made for that particular lock type; if a threshold value is 100, an implicit transaction will begin when the one hundred and first lock is made, etc.

**TTS Get Workstation Thresholds** and **TTS Set Workstation Thresholds** (0x2222 34 8) can be used to change the implicit workstation thresholds and to restore the thresholds.

For example, **TTS Get Workstation Thresholds** can query an application for the number of logical and physical record locks allowed before an implicit transaction begins. **TTS Set Workstation Thresholds** can then do either of the following:

- ♦ Turn off implicit transactions
- ♦ Set implicit thresholds for applications that always keep one or more records locked

Applications that use only explicit transactions, but that sometimes generate unnecessary implicit transactions, should turn off all implicit transactions.

The default thresholds for logical and physical locks is zero. A threshold value of 0xFF means implicit transaction for that lock type is not in effect.

The threshold values are not reset when the application terminates; all applications on the requesting workstation share the same threshold values.

Threshold values are set to zero when End Of Job completes.

DOS workstation users normally use the SETTTS.EXE utility to set the workstation thresholds.

## See Also

**[TTS Set Workstation Thresholds 0x2222 34 08 \(page 1245\)](#)**

## TTS Is Available 0x2222 34 00

Verifies whether the default file server supports transaction tracking.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

### Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (0)	byte

### Return Values

Decimal	Hex	Description
0	0x00	Transaction Tracking Unavailable
253	0xFD	Transaction Tracking Disabled
255	0xFF	Transaction Tracking Available

# TTS Set Application Thresholds 0x2222 34 06

Specifies the number of logical and physical record locks allowed before an implicit transaction begins.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (6)	byte
8	LogicalLockThreshold	byte
9	PhysicalLockThreshold	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful
150	0x96	Server Out Of Memory

## Remarks

If a threshold value is zero, an implicit transaction will begin when the first lock is made for that particular lock type; if a threshold value is 100, an implicit transaction will begin when the one hundred and first lock is made, etc.

**TTS Set Application Thresholds** and **TTS Get Application Thresholds** (0x2222 34 5) can be used to change the implicit application thresholds and to restore the thresholds.

For example, **TTS Get Application Thresholds** can query an application for the number of logical and physical record locks allowed before an implicit transaction begins. **TTS Set Application Thresholds** can then do either of the following:

- ♦ Turn off implicit transactions
- ♦ Set implicit thresholds for applications that always keep one or more records locked

Applications that use only explicit transactions, but that sometimes generate unnecessary implicit transactions, should turn off all implicit transactions.

The default thresholds for logical and physical locks is zero. A threshold value of 0xFF means implicit transaction for that lock type is not in effect.

The returned thresholds are valid only for the requesting application. When the application terminates, the workstation thresholds are restored.

## See Also

[TTS Get Application Thresholds 0x2222 34 05 \(page 1236\)](#)

# TTS Set Transaction Bits 0x2222 34 10

Sets the transaction bits represented by the *ControlFlags* byte.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (10)	byte
8	ControlFlags	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

Currently, only bit 0 is used. If bit 0 of the *ControlFlags* byte is cleared, forced record locking is disabled. To turn forced record locking back on, you must set bit 0 of the *ControlFlags* byte. The default state of forced record locking is enabled. Bits 1 through 6 are currently reserved.

## See Also

**TTS Get Transaction Bits 0x2222 34 09 (page 1238)**



# TTS Set Workstation Thresholds 0x2222 34 08

Specifies the number of logical and physical record locks allowed before an implicit transaction begins.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (8)	byte
8	LogicalLockThreshold	byte
9	PhysicalLockThreshold	byte

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

If a threshold value is zero, an implicit transaction will begin when the first lock is made for that particular lock type; if a threshold value is 100, an implicit transaction will begin when the one hundred and first lock is made, etc.

**TTS Set Workstation Thresholds** and **TTS Get Workstation Thresholds** (0x2222 34 7) can be used to change the implicit workstation thresholds and to restore the thresholds.

For example, **TTS Get Workstation Thresholds** can query an application for the number of logical and physical record locks allowed before an implicit transaction begins. **TTS Set Workstation Thresholds** can then do either of the following:

- ♦ Turn off implicit transactions
- ♦ Set implicit thresholds for applications that always keep one or more records locked

Applications that use only explicit transactions, but that sometimes generate unnecessary implicit transactions, should turn off all implicit transactions.

The default thresholds for logical and physical locks is zero. A threshold value of 0xFF means implicit transaction for that lock type is not in effect.

The threshold values are not reset when the application terminates; all applications on the requesting workstation share the same threshold values.

Threshold values are set to zero when End Of Job completes.

DOS workstation users normally use the SETTTS.EXE utility to set the workstation thresholds.

## See Also

**TTS Get Workstation Thresholds 0x2222 34 07 (page 1239), End Of Job 0x2222 24 (page 187)**

# TTS Transaction Status 0x2222 34 04

Verifies whether a transaction has been written to disk.

**NetWare Servers:** 2.x, 3.x, 4.x, 5.x

## Request Format

Offset	Content	Type
Request header		
6	FunctionCode (34)	byte
7	SubFunctionCode (4)	byte
8	TransactionNumber	long (Hi-Lo)

## Return Values

Decimal	Hex	Description
0	0x00	Successful

## Remarks

*TransactionNumber* is the number returned by TTS End Transaction (0x2222 34 02). This value contains a number assigned and used by the NetWare OS to track each transaction.

Because of the file server caching algorithms, 3-5 seconds (or longer) can elapse before transactions are actually written. Transactions are written to disk in the order in which they are ended.

## See Also

**TTS End Transaction 0x2222 34 02 (page 1234)**



# Revision History

The following table outlines all the changes that have been made to the NCP documentation (in reverse chronological order) since October 1996.

2-28-2005	All sections	Made several edits.
8-31-2004	"File System" on page 275	<p>Added <a href="#">Chapter 22, "Enhanced NCPs," on page 511</a>.</p> <p>Changed reserved request field in <a href="#">Allocate Short Directory Handle 0x2222 87 12 (page 297)</a> to DstNameSpace and added comments about additional AllocateMode bits and their usage.</p> <p>Changed reserved request field to Flags in <a href="#">Set Short Directory Handle 0x2222 87 09 (page 505)</a> and added information about the reply if Flags is set to 0 or 0x40.</p> <p>Changed offset of Data request field of <a href="#">Write to a File 0x2222 87 65 (page 507)</a> from 24 to 22.</p>
10-15-2003	"Connection" on page 175	<p>Added <a href="#">Set Connection Language Encoding 0x2222 23 34 (page 219)</a>.</p> <p>To support new 0x2222 23 34 NCP, also added <a href="#">languageInfo (page 1117)</a> structure (which is used by <a href="#">Enumerate Connection Information from Connection List 0x2222 123 16 (page 978)</a> to retrieve information that the new NCP can set).</p>
10-15-2003	"File System" on page 275	Added new UFT8NCPStringsEnabledBit (0x80) as a possible value for the StatusFlagBits field of <a href="#">VollInfoStructure (page 627)</a> .
4-25-2003	All sections	<p>Adjusted tables to take advantage of a larger PDF page size.</p> <p>Made several improvements to aid in access and usability.</p> <p>Fixed minor typographical error in Revision History list.</p>
1-10-2003	"Statistical" on page 969	<p>Added Remarks about the fields in the FSCounters structure to <a href="#">Get File Server Information 0x2222 123 02 (page 996)</a>.</p> <p>Added Remarks about the user needing to be logged in for <a href="#">User Information 0x2222 123 04 (page 1079)</a> to properly return information.</p>
9-18-2002	"File System" on page 275	Added 0x4 option to <a href="#">Open CallBack Control 0x2222 87 34 (page 394)</a> .
3-29-2002	"File System" on page 275	Updated the values of <a href="#">"OpenCreateMode Values" on page 636</a> .

---

3-18-2002	"Synchronization" on page 1157	<p>Updated the Hi-Lo or Lo-Hi order of several parameters:</p> <ul style="list-style-type: none"> <li>• LockAreaStartOffset and LockAreaLen of <a href="#">Release Physical Record 0x2222 28</a> (page 1206), <a href="#">Clear Physical Record 0x2222 30</a> (page 1170), and <a href="#">Log Physical Record 0x2222 109</a> (page 1191).</li> <li>• LockTimeout of <a href="#">Log File (old) 0x2222 105</a> (page 1185), <a href="#">Lock File Set 0x2222 106</a> (page 1178), <a href="#">Log Logical Record 0x2222 107</a> (page 1187), <a href="#">Lock Logical Record Set 0x2222 108</a> (page 1180), <a href="#">Log Physical Record 0x2222 109</a> (page 1191), and <a href="#">Lock Physical Record Set 0x2222 110</a> (page 1182).</li> <li>• SemaphoreHandle of <a href="#">Open Semaphore (old) 0x2222 32 00</a> (page 1195) and <a href="#">Wait On Semaphore (old) 0x2222 32 02</a> (page 1210).</li> <li>• SemaphoreTimeOut of <a href="#">Wait On (P) Semaphore 0x2222 111 02</a> (page 1211).</li> </ul>
2-20-2002	"File System" on page 275	Updated the field descriptions of <a href="#">VollInfoStructure</a> (page 627).
2-15-2002	"Server Environment" on page 853	Added 64BitOffsetsSupportedFlag to the reply header of <a href="#">Get File Server Information 0x2222 23 17</a> (page 898).
2-15-2002	"File System" on page 275	Added 0x40 value, 64BitFileOffsetsEnabledBit to the StatusFlagBits field of <a href="#">VollInfoStructure</a> (page 627). This bit indicates whether 64-bit offsets are enabled for a given volume. Only NSS volumes support 64-bit offsets.
2-15-2002	"File System" on page 275 and "File System" on page 275	Added information about bit 26 and <a href="#">64BitFileSizeStruct</a> (page 570).
2-15-2002	"File System" on page 275	Added <a href="#">Get Current Size of File 0x2222 87 66</a> (page 328), <a href="#">Read File 0x2222 87 64</a> (page 429), and <a href="#">Write to a File 0x2222 87 65</a> (page 507).
2-15-2002	"Synchronization" on page 1157	Added <a href="#">Clear Physical Record 0x2222 87 69</a> (page 1168), <a href="#">Log Physical Record 0x2222 87 67</a> (page 1189), and <a href="#">Release Physical Record 0x2222 87 68</a> (page 1204).
6-21-2001	<a href="#">printInfo</a> (page 1132) used by <a href="#">Enumerate Connection Information from Connection List 0x2222 123 16</a> (page 978)	Added the <a href="#">printInfo</a> (page 1132) structure and added link to it from <a href="#">Enumerate Connection Information from Connection List 0x2222 123 16</a> (page 978).
6-20-2001	<a href="#">authInfo</a> (page 1089) used by <a href="#">Enumerate Connection Information from Connection List 0x2222 123 16</a> (page 978)	Added a description of the loginStatus field.
4-24-2001	<a href="#">Get File Information 0x2222 87 31</a> (page 347)	Changed the offsets of HandleInfoLevel and NameSpace in the Request Format.
4-20-2001		Updated all documentation for accessibility.

---

---

2-15-2001	"SecretStore" on page 849	Added information about SecretStore NCP 92 and its subverbs.
2-13-2001	Timesync Get Time 0x2222 114 01 (page 1221)	Added information about theTime in the Remarks section.
2-13-2001	"RenameFlag Values" on page 637	Changed RNameOnlyFlag value from 0x3 to 0x4.
2-13-2001	"NCP Numbers" on page 23	Added a section for all NCP return values (including RPC return values).
2-13-2001	RPC Set Set Command Value 0x2222 131 06 (page 844)	Added the SubFuncStrucLen.
2-13-2001	RPC Add Name Space To Volume 0x2222 131 05 (page 834)	Added the SubFuncStrucLen. Also, changed AddNameSpace to AddNameSpaceAndVol. Added a note that the name space should be followed by a space and the volume name (with no colon).
2-13-2001	RPC Unload an NLM 0x2222 131 02 (page 846)	Changed PathAndName to NLMName since no path is necessary to unload the NLM.
2-13-2001	Clear Physical Record 0x2222 30 (page 1170)	Added link to this NCP in numbered list.
2-13-2001	DM Support Module Information 0x2222 90 132 (page 230)	Added info level = 2 information.
2-13-2001	DM File Information 0x2222 90 129 (page 228)	Removed DOSDirectoryEntry and SMediaString and added RestoreTime.
2-13-2001	DMINFO (page 246)	Removed the data stream number field.
2-13-2001	Scan Volume's User Disk Restrictions 0x2222 22 32 (page 476)	Changed the number of entries that can be returned from 12 to 16, and added a better description of how to get additional entries.
2-13-2001	Get Extended Volume Information 0x2222 22 51 (page 345)	Added descriptions.
2-13-2001	VollInfoStructure (page 627)	Added field descriptions.

---

---

2-13-2001	Open/Create File or Subdirectory with Callback 0x2222 87 32 (page 404) and Open/Create File or Subdirectory II with Callback 0x2222 87 33 (page 407)	Added description for OCRetFlags.
2-13-2001	NetWareHandlePathStruct (page 609)	Added field descriptions.
2-13-2001	TotalStreamSizeStruct (page 625)	Added a description for the TtIDSDskSpaceAlloc field.
2-13-2001	Revoke File Handle Rights 0x2222 87 43 (page 448)	Changed the offsets of the Request fields starting with QueryFlag (from 8 to 11) and adjusted the other offsets accordingly.
2-13-2001	Get Extended Volume Information 0x2222 22 51 (page 345)	Changed the SubFuncStructLen from 5 to 2.
2-13-2001	"NetWare Core Protocols Preface" on page 21	Expanded the introductory information in the Preface.
2-13-2001	Obtain File or Subdirectory Information 0x2222 87 06 (page 392)	Added an explanation of SearchAttributes being currently ignored to the Remarks section.
12-14-2000	Get File Server Information 0x2222 23 17 (page 898)	Added LocalLoginInfoCcode, ProductMajorVersion, ProductMinorVersion, ProductRevisionVersion, and OSLanguageID fields to the Reply format.
10-17-2000	"Time Synchronization" on page 1215 and "Time Synchronization" on page 1215	Added time synchronization NCPs and structure.
9-1-2000	Update File Handle Rights 0x2222 87 44 (page 450)	Added documentation for this NCP.
7-12-2000	Read From a File 0x2222 72 (page 431)	Corrected the remarks paragraph explaining how the client must request data blocks. Also, corrected the reference to Negotiate Buffer Size 0x222 33.
5-10-2000	Scan Salvageable File List 0x2222 87 41 (page 470)	Corrected the remarks paragraph explaining how to set the scan sequence variable.

---



---

2-14-2000	Get File Server Date And Time 0x2222 20 (page 895) in Server Environment	Changed the range of values for Second to be 0 to 59. Also, cleaned up the description of DayOfWeek.
2-7-2000	Change Queue Job Entry 0x2222 23 123 (page 760), Change Queue Job Entry (old) 0x2222 23 109 (page 761), Create Queue Job And File 0x2222 23 121 (page 773), Create Queue Job And File (old) 0x2222 23 104 (page 775), Read Queue Job Entry 0x2222 23 122 (page 799), and Read Queue Job Entry (old) 0x2222 23 108 (page 800) in Queue	Changed references to ObjectInfoStruct to <b>JobStruct</b> (page 826).
1-18-2000	FileSystemInfo (page 1104), VollInfoDef (page 1151), and VollInfo2Def (page 1155) in Statistical	Added field definitions for these structures.
1-3-2000	Get Server Set Commands Information By Name 0x2222 123 62 (page 1047) in Statistical	Added documentation for this NCP.
12-20-99	Fragment Close 0x2222 104 03 (page 681) in NDS	Documented this NCP.
11-2-99	Revoke File Handle Rights 0x2222 87 43 (page 448) in File System	Removed return value 255 and changed the former 253 to be Bad Station Number with an accompanying explanation.
11-2-99	"File System" on page 275	Added links to values and expanded " <b>OpenCreateMode Values</b> " on page 636.
10-25-99	Broadcast To Console 0x2222 21 09 (page 646) in Message	Changed the maximum message length from 60 bytes to 255 bytes.

---

---

10-25-99	<a href="#">Send Console Broadcast 0x2222 23 253 (page 962) in Server Environment</a>	Added that this NCP accepts messages up to 250 bytes in length.
10-19-99	<a href="#">Get Media Manager Objects List 0x2222 123 31 (page 1011) in Statistical</a>	Changed the initial value of startNum to from zero to -1.
10-14-99	<a href="#">Get Object Disk Usage and Restrictions 0x2222 22 41 (page 366) in File System</a>	Changed the length of SubFuncStructLen from 5 to 6.
10-8-99	<a href="#">“Extended ReturnInfoMask Values” on page 638 in File System</a>	<p>Changed the structure related to Bit 22 Effective Rights from <a href="#">EAInfoStruct (page 589)</a> to <a href="#">EffectiveRightsStruct (page 590)</a>.</p> <p>Also added paragraph to Extended ReturnInfoMask Values section to explain that the Sibling Count for a subdirectory will always be zero.</p>
10-1-99	<a href="#">LastAccessedTime Struct (page 598) and MacTimeStruct (page 602) in File System</a>	Added field definitions for both structures.
9-28-99	<a href="#">“NCP Numbers” on page 23</a>	<p>Fixed access to several NCPs.</p> <p>Added subfunction table for <a href="#">NCP 104</a>.</p>
9-28-99	<a href="#">Read From a File 0x2222 72 (page 431) and Rename File 0x2222 69 (page 441) in File System</a>	Removed "22" from titles of both NCPs.
9-28-99	<a href="#">“NCP Numbers” on page 23</a>	Deleted extra reference to NCP 31 in NCP By Number table.
9-24-99	<a href="#">Return NCP Extension Maximum Data Size 0x2222 36 06 (page 675) in NCP Extension</a>	Changed the link to this request from the <a href="#">“NCP Numbers” on page 23</a> section. The link for 36 06 used to take you to <a href="#">Get NCP Extension Maximum Data Size 0x2222 36 01 (page 668)</a> .
9-15-99	<a href="#">Get Huge NS Information 0x2222 87 26 (page 353) and Set Huge NS Information 0x2222 87 27 (page 502) in File System</a>	Added descriptions for <i>HugeStateInfo</i> and <i>HugeDataLen</i> .

---

---

9-15-99	ModifyDOSInfoStructure (page 604) in File System	Added a description for <i>MaximumSpace</i> .
9-15-99	Scan Directory Disk Space 0x2222 22 40 (page 455) in File System	Added a description for <i>DataForkFirstFAT</i> and <i>OtherFileForkSize</i> .
8-11-99	Generate Directory Base and Volume Number 0x2222 87 22 (page 326) in File System	Changed the descriptions of <i>dstNameSpace</i> and <i>dstNSIndicator</i> to utilize the 'Jn' designation.  Also, added a paragraph to the Remarks section to further explain the 'Jn' functionality.
8-10-99	Get Connection Using A File 0x2222 23 236 (page 880) in Server Environment	Changed SubFuncStructLen from (5 + PathLen) to (9).  Also, added <a href="#">Convert Path to Dir Entry 0x2222 23 244 (page 301)</a> as another way to obtain the <i>DirectoryBase</i> entry number.
8-10-99	Send Broadcast Message 0x2222 21 10 (page 654) in Message	Added support for NetWare 3.12.
8-6-99	Get Big Packet NCP Max Packet Size 0x2222 97 (page 189) in Connection	Updated the explanation of <i>SecurityFlag</i> in the Remarks section.
8-6-99	Scan Bindery Object (List) 0x2222 23 32 (page 162) in Bindery	Changed <i>InfoFlags</i> from Lo-Lo to Lo-Hi.  Changed offset 23 to 23 + ObjectNameLen.
7-28-99	MLID Board Information 0x2222 123 27 (page 1071) in Statistical	Changed the description of offset 15. For NetWare 4.x, this offset is <i>reserved</i> . For NetWare 5.x, this offset is <i>numberOfProtocols</i> .  Also, added a statement about the differences between NetWare 4.x and NetWare 5.x in the Remarks section.
7-28-99	Modify File or Subdirectory DOS Information 0x2222 87 07 (page 389) in File System	Changed the offsets of <i>ModifyDOSInfoStruct</i> and <i>NWHandlePathStruct</i> from 44 to 16 and from 82 to 54 respectively.
7-28-99	All NCPs	Added NetWare 5.x support to all NCPs that previously supported NetWare 4.x.
7-7-99	Map Directory Number to Path 0x2222 23 243 (page 385) in File System	Removed statement about <i>Path</i> being NULL-terminated. Also, added an example of how <i>Path</i> uses a length-preceded string to return a directory path.

---

---

6-18-99	Rename Or Move a File or Subdirectory 0x2222 87 04 (page 445) in File System	Added bit definitions and descriptions for <i>RenameFlag</i> .
6-17-99	Get Internet Address 0x2222 23 26 (page 192) in Connection	Added connection type 11 UDP (for IP) as well as a description of how this NCP behaves when called from an IP-only server.
6-16-99	RPC Load an NLM 0x2222 131 01 (page 840) in RPC	Deleted 158 Bad File Name or No File Name Given as a possible return value.
6-16-99	NetWareInformationStruct (page 611) in File System	Added descriptions of the date and time values.
5-24-99	Enumerate NCP Service Network Addresses 0x2222 123 17 (page 981) in Statistical	Changed offset 14 in the reply buffer from <i>reserved</i> to <i>serverInfoFlags</i> . Also, added 0x00000001 CLUSTER_MEMBER_BIT.
5-5-99	LAN Custom Counters Information 0x2222 123 23 (page 1063) in Statistical	Changed <i>MoreFlag</i> to <i>Unused</i> . Also, added explanation on how to ensure all information has been received by using <i>StartNumber</i> and <i>NumOfCCinPkt</i> .
5-3-99	Commit File 0x2222 59 (page 300) in File System	Changed <i>Reserved</i> from 3 bytes to 1.
4-8-99	"File Attributes" on page 277 in File System	Changed Byte 1, Bit 13 to Reserved.
3-12-99	Add Trustee Set to File or Subdirectory 0x2222 87 10 (page 289), Delete Trustee Set from File or SubDirectory 0x2222 87 11 (page 316) in File System	Added comments about <i>NWHandlePathStruct</i> .
3-11-99	Get Server Set Categories 0x2222 123 61 (page 1043) in Statistical	Reversed the order of <i>NumberOfSetCategories</i> and <i>NextStartNumber</i> . Also, removed <i>CategoryNameLen</i> .
3-11-99	Revoke File Handle Rights 0x2222 87 43 (page 448) in File System	Added "(Hi-Lo)" to <i>FileHandle</i> in both the request and reply buffers and added a comment about these fields.

---

---

3-11-99	Open CallBack Control 0x2222 87 34 (page 394) in File System	Added "(Hi-Lo)" to <i>CCFileHandle</i> and added a comment about this field.
3-11-99	Open/Create File or Subdirectory 0x2222 87 01 (page 398) in File System	Changed <i>FileHandle</i> from 6 bytes to a long.
2-19-99	Scan File or Subdirectory for Trustees 0x2222 87 05 (page 468) in File System	Changed offset for <i>NwHandlePathStruct</i> from 14 to 16.
2-18-99	"File Attributes" on page 277 in File System	Added the Subdirectories bit (Bit 4) and Bit 4 to Bit 5.
2-17-99	Scan Bindery Object (List) 0x2222 23 32 (page 162) in Bindery	Added documentation for this new NCP.
2-11-99	Get Directory Disk Space Restriction 0x2222 87 39 (page 334), Search for File or Subdirectory Set (Extended Errors) 0x2222 87 40 (page 484), Scan Salvageable File List 0x2222 87 41 (page 470), Purge Salvageable File List 0x2222 87 42 (page 425) in File System	Added documentation for four new NCPs.
2-9-99	Log File 0x2222 87 36 (page 1184) in Synchronization	Changed <i>SubFunctionCode</i> from 33 to 36.
1-25-99	Enumerate Connection Information from Connection List 0x2222 123 16 (page 978) in Statistical	Added descriptions for the connection information that is returned.
1-13-99	Get File Server Information 0x2222 23 17 (page 898) in Server Environment	Changed the description of <i>MaximumServiceConnections</i> and <i>MaxConnectionsEverUsed</i> .

---

---

1-11-99	Get Extended Volume Information 0x2222 22 51 (page 345) in File System	Changed <i>VolumeNumber</i> from a LONG to a BYTE.
11-13-98	Revoke File Handle Rights 0x2222 87 43 (page 448) in File System	Added documentation for this new NCP.
11-3-98	Search for File or Subdirectory Set 0x2222 87 20 (page 482) in File System	Changed <i>MoreEntriesFlag</i> description so that 0xFF indicates that more entries are available, rather than 1.
11-3-98	CPU Information 0x2222 123 08 (page 976) in Statistical	Added <i>reserved</i> to the reply buffer.  Also, changed <i>NumberOfCPUs</i> from a WORD to a LONG. Removed comment that <i>NumberOfCPUs</i> was always set to 1.
6-26-98	“Extended ReturnInfoMask Values” on page 638 in File System	Added Bit 24, Last Accessed Time to the possible values.
6-10-98	Get File Server Information 0x2222 23 17 (page 898) in Server Environment	Changed offset 76 in the reply buffer from <i>reserved</i> to <i>MixedModePathFlag</i> .
3-19-98	Execute NCP Extension 0x2222 37 (page 664) in NCP Extension	Updated the request and reply buffers.
3-4-98	Enumerate NCP Service Network Addresses 0x2222 123 17 (page 981) in Statistical	Added <i>serverGUID</i> field to the reply buffer.
2-27-98	Get NLM Loaded List 0x2222 123 10 (page 1020) in Statistical	Changed <i>More</i> to <i>unused</i> in the reply buffer.  Also, added <i>NLMcount</i> and <i>NLMsInList</i> .
2-27-98	Generate GUIDs 0x2222 23 33 (page 188) in Connection	Added documentation for a new NCP that returns a list of GUIDs.
2-13-98	Enumerate NCP Service Network Addresses 0x2222 123 17 (page 981) in Statistical	Changed the length of <i>SubFuncStrucLen</i> from 6 to 5.

---

---

2-6-98	Get Connection's Open Files 0x2222 23 235 (page 866) in Server Environment	Changed <i>ParentDirectoryNumber</i> and <i>DirectoryNumber</i> to <i>DOSParentDirectoryNumber</i> and <i>DOSDirectoryNumber</i> respectively.
1-20-98	Connection Message Control 0x2222 21 12 (page 647) in Message	Added documentation for a new NCP that controls broadcast and watchdog messages.
10-28-97	Enumerate Connection Information from Connection List 0x2222 123 16 (page 978), Enumerate NCP Service Network Addresses 0x2222 123 17 (page 981) in Statistical	Added documentation for two new NCPs that enumerate information.
9-18-97	Send NDS Fragmented Request/Reply 0x2222 104 02 (page 688) in NDS	Switched the order of <i>Flags</i> and <i>Verb</i> in the request buffer.
9-16-97	Get Server Set Commands Information 0x2222 123 60 (page 1045) in Statistical	Changed <i>SetCmdFlags</i> and <i>SetCmdName</i> from length-preceded strings to NULL-terminated strings.
6-3-97	Return NCP Extension Maximum Data Size 0x2222 36 06 (page 675) in NCP Extension	Added <i>SubFuncStrucLen</i> to the request buffer. Also, deleted <i>NCPExtensionNumber</i> from the request buffer.
6-3-97	Get NCP Extension Registered Verbs List 0x2222 36 04 (page 672) in NCP Extension	Added <i>SubFuncStrucLen</i> to the request buffer. Also, added a note that <i>StartingNumber</i> should be zero for the initial request.
6-3-97	Get Number of Registered NCP Extensions 0x2222 36 03 (page 671) in NCP Extension	Added <i>SubFuncStrucLen</i> to the request buffer.
6-3-97	Get NCP Extension Information by Name 0x2222 36 02 (page 669) in NCP Extension	Added <i>SubFuncStrucLen</i> to the request buffer. Also, added <i>NCPextensionCustomData</i> to the reply buffer.

---

---

6-3-97	Get NCP Extension Maximum Data Size 0x2222 36 01 (page 668) in NCP Extension	Added <i>SubFuncStrucLen</i> to the request buffer.  Also, deleted <i>NCPEExtensionName</i> and <i>NCPEExtensionLength</i> from the request buffer.
6-3-97	Get NCP Extension Information (old) 0x2222 36 00 (page 666), Return NCP Extension Information 0x2222 36 05 (page 673) in NCP Extension	Added <i>SubFuncStrucLen</i> and <i>NCPEExtensionNumber</i> to the request buffer.  Also, added <i>NCPEextensionCustomData</i> to the reply buffer.
6-3-97	Change Connection State 0x2222 23 29 (page 180) in Connection	Added 0xE0 ERR_NO_LOGIN_CONNECTIONS_AVAILABLE return value.
3-18-97	Open/Create File or Subdirectory 0x2222 87 01 (page 398), Open/Create File or Subdirectory 0x2222 87 30 (page 401), Open/Create File or Subdirectory with Callback 0x2222 87 32 (page 404), Open/Create File or Subdirectory II with Callback 0x2222 87 33 (page 407) in File System	Added a description of <i>DesiredAccessRights</i> when used to create a subdirectory.
3-12-97	Map Directory Number to Path 0x2222 23 243 (page 385) in File System	Added a description of <i>Path</i> .
2-28-97	Send NDS Fragmented Request/Reply 0x2222 104 02 (page 688) in NDS	Changed the offset for <i>ReplyData</i> from 14 to 16.
2-13-97	LAN Name Information 0x2222 123 24 (page 1065) in Statistical	Documented the <i>BoardName</i> structure.
2-13-97	MLIDBoardInfo (page 1123) in Statistical	Added descriptions for the <i>ProtocolID</i> and <i>ProtocolName</i> fields.

---



---

2-4-97	Get File Information 0x2222 87 31 (page 347) in File System	Added all levels of information that can be returned.
12-17-99	Get Cache Information 0x2222 123 01 (page 987) in Statistical	Documented additional information passed back if <i>SubFuncStructLen</i> = 3 and <i>VersionNumber</i> = 1.
12-16-96	Remove User Disk Space Restrictions 0x2222 22 34 (page 438) in File System	Changed the length of <i>SubFuncStructLen</i> from 5 to 6.
12-16-96	Get File Server Login Status 0x2222 23 205 (page 907) in Server Environment	Added that a user without console privileges can make this request.  Also, changed the No Privileges return value to Unknown Request.
12-16-96	KnownServStruc (page 1110) in Statistical	Changed the <i>ServerName</i> string from length-preceeded to NULL-terminated.
11-18-96	MLIDBoardInfo (page 1123) in Statistical	Added <i>numberOfProtocols</i> to the reply buffer.  Changed <i>reserved</i> from a WORD to a BYTE.
10-29-96	Get Effective Rights for Directory Entry 0x2222 22 42 (page 344) in File System	Changed the length of <i>SubFuncStructLen</i> from 2+ to 3+PathLen.
10-29-96	ReferenceIDStruct (page 620) in File System	Changed the LONG field type to a WORD field type.
10-29-96	Volume DM Status 0x2222 90 130 (page 243) in Data Migration	Documented four additional return fields.
10-24-96	Convert Path to Dir Entry 0x2222 23 244 (page 301) in File System	Changed the type of <i>DirectoryHandle</i> from a LONG field to a BYTE field.

---

