NetView for AIX
Programmer's Reference
Version 4

Document Number SC31-8165-00

April 8, 1996

NetView for AIX

SC31-8165-00

Programmer's Reference

Version 4

NetView for AIX

SC31-8165-00

Programmer's Reference

Version 4

Note

Before using this product, read the general information under “Notices” on page xv.

First Edition (July 1995)

This document applies to IBM NetView for AIX (feature 5608), which is a feature of SystemView for AIX (5765-527). IBM NetView for AIX runs under the AIX Operating System for RISC System/6000 Version 3 Release 2 (5756-030) or Version 4 Release 1 (5765-393). This product is based, in part, on Hewlett-Packard Company's OpenView product.

Publications are not stocked at the address given below. If you want more IBM publications, ask your IBM representative or write to the IBM branch office serving your locality.

A form for your comments is provided at the back of this document. If the form has been removed, you may address comments to:

IBM Corporation
Department CGMD
P.O. Box 12195
Research Triangle Park, North Carolina 27709
U.S.A.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Notices . . . xv
Trademarks . . . xv

About This Book . . . xvii
Who Should Use This Book . . . xvii
How to Use This Book . . . xvii
Highlighting and Operation Naming Conventions . . . xvii
Format of Reference Pages . . . xviii
Meaning of Function Numbers . . . xviii
Where to Find More Information . . . xix

Chapter 1. Function Tables for NetView for AIX Man Pages . . . 1
Graphical User Interface Routines . . . 1
SNMP Routines . . . 12
WinSNMP Functions . . . 14
XMP Functions . . . 16
Filtering and Thresholding Functions . . . 18
GTM API Routines . . . 18
Collection Facility Routines . . . 21
Client/Server APIs . . . 22
Security Functions . . . 22
Miscellaneous Functions . . . 23
Introductions . . . 23

Chapter 2. Reference Pages . . . 25
at_array_to_oid(3) . . . 26
at_free(3) . . . 27
at_oid_match(3) . . . 28
at_oid_to_array(3) . . . 29
at_oid_to_str(3) . . . 30
at_str_to_oid(3) . . . 31
gtmdump(8) . . . 32
mp_abandon(3) . . . 34
mp_action_req(3) . . . 36
mp_action_rsp(3) . . . 39
mp_bind(3) . . . 42
mp_cancel_get_req(3) . . . 44
mp_cancel_get_rsp(3) . . . 47
mp_create_req(3) . . . 49
mp_create_rsp(3) . . . 52
mp_delete_req(3) . . . 55
mp_delete_rsp(3) . . . 58
mp_error_message(3) . . . 61
mp_event_report_req(3) . . . 62
mp_event_report_rsp(3) . . . 65
mp_get_next_req(3) . . . 67
mp_get_req(3) . . . 69
mp_get_rsp(3) . . . 72
mp_initialize(3) . . . 75
mp_receive(3) . . . 76

mp_set_req(3) . . . 81
mp_set_rsp(3) . . . 84
mp_shutdown(3) . . . 87
mp_unbind(3) . . . 88
mp_version(3) . . . 90
mp_wait(3) . . . 93
nvCollectionAdd(3) . . . 95
nvCollectionAddCallback(3) . . . 98
nvCollectionDelete(3) . . . 100
nvCollectionDone(3) . . . 102
nvCollectionError(3) . . . 103
nvCollectionErrorMsg(3) . . . 104
nvCollectionEvaluate(3) . . . 105
nvCollectionFreeDefn(3) . . . 107
nvCollectionGetAllForObject(3) . . . 109
nvCollectionGetInfo(3) . . . 111
nvCollectionGetTimestamp(3) . . . 113
nvCollectionIntersect(3) . . . 114
nvCollectionListCollections(3) . . . 116
nvCollectionModify(3) . . . 117
nvCollectionOpen(3) . . . 119
nvCollectionRead(3) . . . 121
nvCollectionResolve(3) . . . 122
nvCollectionUnion(3) . . . 124
nvFilterDefine(3) . . . 126
nvFilterDelete(3) . . . 129
nvFilterErrorMsg(3) . . . 130
nvFilterFreeNameList(3) . . . 131
nvFilterGet(3) . . . 132
nvFilterGetNameList(3) . . . 134
NVisClient(3) . . . 136
nvotChangeArcDetails(3) . . . 137
nvotChangeArcIconInGraph(3) . . . 141
nvotChangeArcLabelInGraph(3) . . . 146
nvotChangeArcStatus(3) . . . 150
nvotChangeBoxBackground(3) . . . 154
nvotChangeBoxDetails(3) . . . 157
nvotChangeBoxIconInGraph(3) . . . 160
nvotChangeBoxLabelInGraph(3) . . . 163
nvotChangeBoxPositionInGraph(3) . . . 166
nvotChangeGraphBackground(3) . . . 169
nvotChangeGraphDetails(3) . . . 172
nvotChangeGraphIcon(3) . . . 175
nvotChangeGraphIconInGraph(3) . . . 178
nvotChangeGraphLabelInGraph(3) . . . 181
nvotChangeGraphPositionInGraph(3) . . . 184
nvotChangeRootGraphIcon(3) . . . 187
nvotChangeRootGraphLabel(3) . . . 190
nvotChangeUnderlyingArcIcon(3) . . . 193
nvotChangeVertexDetails(3) . . . 197
nvotChangeVertexIconInBox(3) . . . 200
nvotChangeVertexIconInGraph(3) . . . 203
nvotChangeVertexLabelInBox(3) . . . 206
nvotChangeVertexLabelInGraph(3) . . . 209

nvotChangeVertexPositionInBox(3) . . . 212
nvotChangeVertexPositionInGraph(3) . . . 215
nvotChangeVertexStatus(3) . . . 218
nvotCreateArcInGraph(3) . . . 221
nvotCreateBoxInGraph(3) . . . 227
nvotCreateGraph(3) . . . 232
nvotCreateGraphInGraph(3) . . . 235
nvotCreateParallelUnderlyingArc(3) . . . 240
nvotCreateProvidingSap(3) . . . 245
nvotCreateRootGraph(3) . . . 249
nvotCreateSerialUnderlyingArc(3) . . . 253
nvotCreateUsingSap(3) . . . 258
nvotCreateVertexInBox(3) . . . 261
nvotCreateVertexInGraph(3) . . . 265
nvotDeleteArc(3) . . . 269
nvotDeleteArcFromGraph(3) . . . 272
nvotDeleteBox(3) . . . 276
nvotDeleteBoxFromGraph(3) . . . 278
nvotDeleteGraph(3) . . . 281
nvotDeleteGraphFromGraph(3) . . . 283
nvotDeleteProvidingSap(3) . . . 286
nvotDeleteUnderlyingArc(3) . . . 289
nvotDeleteUsingSap(3) . . . 292
nvotDeleteVertex(3) . . . 295
nvotDeleteVertexFromBox(3) . . . 297
nvotDeleteVertexFromGraph(3) . . . 299
nvotDone(3) . . . 302
nvotFree(3) . . . 304
nvotGetArcsInGraph(3) . . . 307
nvotGetArcObjectId(3) . . . 310
nvotGetBoxesInGraph(3) . . . 314
nvotGetBoxObjectId(3) . . . 317
nvotGetBoxesWhichVertexIsMemberOf(3) . . . 320
nvotGetError(3) . . . 323
nvotGetErrorMsg(3) . . . 326
nvotGetGraphObjectId(3) . . . 327
nvotGetGraphsInGraph(3) . . . 330
nvotGetGraphsWhichArcIsMemberOf(3) . . . 333
nvotGetGraphsWhichBoxIsMemberOf(3) . . . 338
nvotGetGraphsWhichGraphIsMemberOf(3) . . . 341
nvotGetGraphsWhichVertexIsMemberOf(3) . . . 344
nvotGetSapsOnVertex(3) . . . 347
nvotGetVertexObjectId(3) . . . 350
nvotGetVerticesInBox(3) . . . 353
nvotGetVerticesInGraph(3) . . . 356
nvotInit(3) . . . 359
nvotSetCenterBoxForGraph(3) . . . 362
nvotSetCenterGraphForGraph(3) . . . 365
nvotSetSynchronousCreation(3) . . . 368
nvotVertexHandler(3) . . . 370
nvSnmpBlockingGetTable(3) . . . 376
nvSnmpErrString(3) . . . 379
nvSnmpTrapOpenFilter(3) . . . 380
nvs_Audit(3) . . . 384

nvs_deleteSecContext(3) . . . 386
nvs_getClientPerms(3) . . . 388
nvs_isClientAuthorized(3) . . . 391
nvs_SecErrMsg(3) . . . 394
nvs_isSecOn(3) . . . 395
om_copy(3) . . . 396
om_copy_value(3) . . . 398
om_create(3) . . . 400
om_decode(3) . . . 402
om_delete(3) . . . 404
om_encode(3) . . . 406
om_get(3) . . . 408
om_instance(3) . . . 412
om_put(3) . . . 414
om_read(3) . . . 417
om_remove(3) . . . 419
om_write(3) . . . 421
OVDefaultServerName(3) . . . 424
OVeDeregister(3) . . . 425
OVeFilterAttr(3) . . . 427
OVeRegister(3) . . . 431
OVmib_get_objid_name(3) . . . 433
OVmib_read_objid(3) . . . 434
OVsnmpAddVarBind(3) . . . 435
OVsnmpBlockingSend(3) . . . 437
OVsnmpClose(3) . . . 440
OVsnmpConfAllocEntry(3) . . . 442
OVsnmpConfAllocWcList(3) . . . 443
OVsnmpConfClose(3) . . . 444
OVsnmpConfCopyEntry(3) . . . 445
OVsnmpConfCreateEntry(3) . . . 446
OVsnmpConfDbName(3) . . . 448
OVsnmpConfDeleteCache(3) . . . 449
OVsnmpConfDeleteEntry(3) . . . 450
OVsnmpConfExportFile(3) . . . 452
OVsnmpConfFileName(3) . . . 454
OVsnmpConfFreeDest(3) . . . 456
OVsnmpConfFreeEntry(3) . . . 457
OVsnmpConfFreeWcList(3) . . . 459
OVsnmpConfOpen(3) . . . 460
OVsnmpConfImportFile(3) . . . 463
OVsnmpConfParseEntry(3) . . . 465
OVsnmpConfPrintCntl(3) . . . 467
OVsnmpConfPrintDest(3) . . . 468
OVsnmpConfPrintEntry(3) . . . 469
OVsnmpConfReadCntl(3) . . . 470
OVsnmpConfReadDefault(3) . . . 472
OVsnmpConfReadEntry(3) . . . 474
OVsnmpConfReadNextDest(3) . . . 476
OVsnmpConfReadNextEntry(3) . . . 478
OVsnmpConfReadWcList(3) . . . 480
OVsnmpConfResolveDest(3) . . . 482
OVsnmpConfStoreCntl(3) . . . 484
OVsnmpConfStoreDefault(3) . . . 486

OVsnmpConfStoreEntry(3) . . . 488
OVsnmpCreatePdu(3) . . . 490
OVsnmpDoRetry(3) . . . 492
OVsnmpErrString(3) . . . 494
OVsnmpFixPdu(3) . . . 495
OVsnmpFreePdu(3) . . . 497
OVsnmpGetRetryInfo(3) . . . 499
OVsnmpIntro(5) . . . 501
OVsnmpOpen(3) . . . 507
OVsnmpRead(3) . . . 510
OVsnmpRecv(3) . . . 512
OVsnmpSend(3) . . . 514
OVsnmpTrapOpen(3) . . . 517
OVsPMD_API(3) . . . 520
OVuTL(3) . . . 522
OVwAckMapClose(3) . . . 528
OVwAckUserSubmapCreate(3) . . . 530
OVwAddActionCallback(3) . . . 532
OVwAddAlertCallback(3) . . . 535
OVwAddCallback(3) . . . 539
OVwAddHelpCallback(3) . . . 541
OVwAddInput(3) . . . 543
OVwAddMenuItem(3) . . . 545
OVwAddMenuItemFunction(3) . . . 547
OVwAddObjMenuItem(3) . . . 550
OVwAddObjMenuItemFunction(3) . . . 552
OVwAddToolPalItem(3) . . . 555
OVwAlertMsg(3) . . . 558
OVwApiIntro(5) . . . 560
OVwBeginMapSync(3) . . . 573
OVwCheckAction(3) . . . 575
OVwConfirmAcknowledgeObjectsCB(3) . . . 578
OVwConfirmCapabilityChangeCB(3) . . . 580
OVwConfirmCreateObjectsCB(3) . . . 582
OVwConfirmCreateSubmapsCB(3) . . . 584
OVwConfirmCreateSymbolsCB(3) . . . 586
OVwConfirmDeleteObjectsCB(3) . . . 588
OVwConfirmDeleteSubmapsCB(3) . . . 590
OVwConfirmExplodeObjectCB(3) . . . 592
OVwConfirmHideSymbolsCB(3) . . . 594
OVwConfirmManageObjectsCB(3) . . . 596
OVwConfirmMoveSymbolCB(3) . . . 599
OVwConfirmObjectStatusCB(3) . . . 601
OVwCreateAction(3) . . . 603
OVwCreateApp(3) . . . 607
OVwCreateMenu(3) . . . 611
OVwCreateMenuItem(3) . . . 613
OVwCreateObjMenuItem(3) . . . 616
OVwCreateSubmap(3) . . . 619
OVwCreateSymbol(3) . . . 623
OVwDbAppendEnumConstants(3) . . . 633
OVwDbCreateField(3) . . . 635
OVwDbCreateObject(3) . . . 638
OVwDbDeleteObject(3) . . . 641

OVwDbFieldNameToFieldId(3) . . . 643
OVwDbGetEnumConstants(3) . . . 645
OVwDbGetFieldInfo(3) . . . 648
OVwDbGetFieldValue(3) . . . 650
OVwDbGetFieldValues(3) . . . 654
OVwDbGetFieldValuesByObjects(3) . . . 656
OVwDbGetUniqObjectName(3) . . . 658
OVwDbHostnameToObjectId(3) . . . 660
OVwDbInit(3) . . . 662
OVwDbListFields(3) . . . 664
OVwDbListObjectsByFieldValue(3) . . . 667
OVwDbNameToObjectId(3) . . . 670
OVwDbSelectionNameToObjectId(3) . . . 672
OVwDbSetEnumConstants(3) . . . 674
OVwDbSetFieldValue(3) . . . 676
OVwDbSetSelectionName(3) . . . 679
OVwDbUnsetFieldValue(3) . . . 681
OVwDisplaySubmap(3) . . . 683
OVwDone(3) . . . 685
OVwEndSessionCB(3) . . . 686
OVwError(3) . . . 688
OVwErrorMsg(3) . . . 689
OVwEventIntro(5) . . . 691
OVwFileDescriptor(3) . . . 694
OVwFindMenuItem(3) . . . 696
OVwGetAppConfigValues(3) . . . 698
OVwGetAppName(3) . . . 701
OVwGetConnSymbol(3) . . . 702
OVwGetFirstAction(3) . . . 706
OVwGetFirstMenuItem(3) . . . 708
OVwGetFirstMenuItemFunction(3) . . . 710
OVwGetFirstObjMenuItem(3) . . . 712
OVwGetFirstObjMenuItemFunction(3) . . . 715
OVwGetFirstRegContext(3) . . . 717
OVwGetMapInfo(3) . . . 719
OVwGetMenuItemPath(3) . . . 721
OVwGetMenuPathSeparator(3) . . . 723
OVwGetObjectInfo(3) . . . 725
OVwGetObjectMenuItemPath(3) . . . 727
OVwGetRegContext(3) . . . 729
OVwGetSelections(3) . . . 731
OVwGetSubmapInfo(3) . . . 733
OVwGetSymbolInfo(3) . . . 735
OVwGetSymbolsByObject(3) . . . 737
OVwHighlightObject(3) . . . 739
OVwInit(3) . . . 741
OVwIsIdNull(3) . . . 743
OVwListObjectsOnMap(3) . . . 745
OVwListSubmaps(3) . . . 747
OVwListSymbols(3) . . . 750
OVwListSymbolTypes(3) . . . 753
OVwLockRegUpdates(3) . . . 755
OVwMainLoop(3) . . . 757
OVwMapCloseCB(3) . . . 759

OVwMapOpenCB(3) . . . 761
OVwPeekOVwEvent(3) . . . 763
OVwPending(3) . . . 765
OVwProcessEvent(3) . . . 767
OVwRegIntro(5) . . . 769
OVwRenameRegContext(3) . . . 795
OVwSaveRegUpdates(3) . . . 797
OVwSelectListChangeCB(3) . . . 799
OVwSetBackgroundGraphic(3) . . . 801
OVwSetStatusOnObject(3) . . . 803
OVwSetSubmapName(3) . . . 806
OVwSetSymbolApp(3) . . . 808
OVwSetSymbolBehavior(3) . . . 810
OVwSetSymbolLabel(3) . . . 813
OVwSetSymbolPosition(3) . . . 815
OVwSetSymbolStatusSource(3) . . . 820
OVwSetSymbolType(3) . . . 822
OVwShowHelp(3) . . . 825
OVwSubmapCloseCB(3) . . . 827
OVwSubmapOpenCB(3) . . . 829
OVwUserSubmapCreateCB(3) . . . 831
OVwVerifyAdd(3) . . . 833
OVwVerifyAppConfigChange(3) . . . 839
OVwVerifyConnect(3) . . . 842
OVwVerifyDeleteSymbol(3) . . . 846
OVwVerifyDescribeChange(3) . . . 849
OVwXtAddInput(3) . . . 852
OVwXtMainLoop(3) . . . 855
SnmpCleanup(3) . . . 858
SnmpClose(3) . . . 860
SnmpContextToStr(3) . . . 862
SnmpCountVbl(3) . . . 865
SnmpCreatePdu(3) . . . 867
SnmpCreateSession(3) . . . 869
SnmpCreateVbl(3) . . . 872
SnmpDecodeMsg(3) . . . 874
SnmpDeleteVb(3) . . . 876
SnmpDuplicatePdu(3) . . . 878
SnmpDuplicateVbl(3) . . . 880
SnmpEncodeMsg(3) . . . 882
SnmpEntityToStr(3) . . . 884
SnmpFreeContext(3) . . . 886
SnmpFreeDescriptor(3) . . . 888
SnmpFreeEntity(3) . . . 890
SnmpFreePdu(3) . . . 892
SnmpFreeVbl(3) . . . 894
SnmpGetLastError(3) . . . 896
SnmpGetLastErrorStr(3) . . . 898
SnmpGetPduData(3) . . . 900
SnmpGetRetransmitMode(3) . . . 903
SnmpGetRetry(3) . . . 905
SnmpGetTimeout(3) . . . 907
SnmpGetTranslateMode(3) . . . 909
SnmpGetVb(3) . . . 911

SnmpOidCompare(3) . . . 913
SnmpOidCopy(3) . . . 915
SnmpOidToStr(3) . . . 917
SnmpRecvMsg(3) . . . 919
SnmpRegister(3) . . . 922
SnmpSelect(3) . . . 925
SnmpSendMsg(3) . . . 927
SnmpSetPduData(3) . . . 930
SnmpSetRetransmitMode(3) . . . 932
SnmpSetRetry(3) . . . 934
SnmpSetTimeout(3) . . . 936
SnmpSetTranslateMode(3) . . . 938
SnmpSetVb(3) . . . 940
SnmpStartup(3) . . . 942
SnmpStrToContext(3) . . . 945
SnmpStrToEntity(3) . . . 948
SnmpStrToOid(3) . . . 951
XnvApplicationShell(3) . . . 953
XnvTopLevelShell(3) . . . 956

Chapter 3. XOM Package . . . 959
Class Hierarchy . . . 959
Class Definitions . . . 959

Chapter 4. XMP API Management Service Packages . . . 963
General Information . . . 963
OM Class Hierarchies . . . 963
The OM Classes . . . 969

Chapter 5. XMP API Management Contents Packages . . . 1033
LNV Package Object Identifier . . . 1033
DMI Package Object Identifier . . . 1033

Chapter 6. Using NetView for AIX GTM Data Structures . . . 1061
Basic Structures . . . 1061
Table Structures . . . 1064
Type Structures . . . 1068

Glossary and Bibliography . . . 1077

Glossary . . . 1079

Bibliography . . . 1107
NetView for AIX Publications . . . 1107
IBM RISC System/6000 Publications . . . 1107
NetView Publications . . . 1108
TCP/IP Publications for AIX (RS/6000, PS/2, RT, 370) . . . 1108
AIX SNA Services/6000 Publications . . . 1108
Internet Request for Comments (RFCs) . . . 1108
Related Publications . . . 1109

Tables

1. Graphical User Interface Routines and Their Reference Pages . . . 1
2. SNMP Routines and Their Reference Pages . . . 12
3. WinSNMP Functions and Their Reference Pages . . . 15
4. XMP Functions and Their Reference Pages . . . 16
5. Filtering and Thresholding Functions and Their Reference Pages . . . 18
6. GTM API Routines and Their Reference Pages . . . 18
7. Collection Facility Routines and Their Reference Pages . . . 21
8. Client/Server APIs and Their Reference Pages . . . 22
9. Security Functions and Their Reference Pages . . . 22
10. Miscellaneous Functions and Their Reference Pages . . . 23
11. API Introductions and Their Reference Pages . . . 23
12. Service Primitives . . . 76
13. Validity of Completion Flag Values . . . 77
14. Valid CMIS-Service-Error Values for each Confirm Primitive . . . 78
15. Bitmask Permissions for nvs_getClientPerms API . . . 388
16. Output of logging INFORMATIVE messages in the OVEXTERNAL subsystem . . . 526
17. Output of Tracing in OVEXTERNAL Subsystem . . . 527
18. Return Codes . . . 563
19. EUI API Events and Their Callbacks . . . 691
20. Widget Resources for XnvApplicationShell . . . 954
21. Widget Resources for XnvTopLevelShell . . . 957
22. General Information about the XOM Package . . . 959
23. Attributes Specific to Encoding . . . 960
24. Attributes Specific to External . . . 960
25. Attributes Specific to Object . . . 961
26. General Information about the Management Service Packages . . . 963
27. Hierarchical Organization of Management Service OM Classes . . . 964
28. OM Attributes of an Action-Error Object . . . 971
29. OM Attributes of an Action-Error-Info Object . . . 972
30. OM Attributes of an Action-Info Object . . . 973
31. OM Attributes of an Action-Reply Object . . . 973
32. OM Attributes of an Action-Type-Id Object . . . 974
33. OM Attributes of an AE-Title . . . 975
34. OM Attributes of an Application-Syntax Object . . . 975
35. OM Attributes of an Attribute Object . . . 976
36. OM Attributes of an Attribute-Error Object . . . 976
37. OM Attributes of an Attribute-Id Object . . . 978
38. OM Attributes of an Attribute-Id-Error Object . . . 978
39. OM Attributes of an Attribute-Id-List Object . . . 979
40. OM Attributes of an AVA Object . . . 979
41. OM Attributes of a Base-Managed-Object-Id Object . . . 980
42. OM Attributes of a CMIS-Action-Argument Object . . . 981
43. OM Attributes of a CMIS-Action-Result Object . . . 982
44. OM Attribute of a CMIS-Cancel-Get-Argument Object . . . 982
45. OM Attributes of a CMIS-Create-Argument Object . . . 983
46. OM Attributes of a CMIS-Create-Result Object . . . 984
47. OM Attributes of a CMIS-Delete-Argument Object . . . 984
48. OM Attributes of a CMIS-Delete-Result Object . . . 985
49. OM Attributes of a CMIS-Event-Report-Argument Object . . . 986
50. OM Attributes of a CMIS-Event-Report-Result Object . . . 987

51. OM Attributes of a CMIS-Filter Object . . . 988
52. OM Attributes of a CMIS-Get-Argument Object . . . 988
53. OM Attributes of a CMIS-Get-List-Error Object . . . 990
54. OM Attributes of a CMIS-Get-Result Object . . . 990
55. OM Attributes of a CMIS-Linked-Reply-Argument Object . . . 991
56. OM Attributes of a CMIS-Service-Error Object . . . 992
57. Problem and Parameter Values for a CMIS-Service-Error Object . . . 992
58. OM Attributes of a CMIS-Set-Argument Object . . . 994
59. OM Attributes of a CMIS-Set-List-Error Object . . . 996
60. OM Attributes of a CMIS-Set-Result Object . . . 996
61. OM Attributes of a Communications-Error Object . . . 997
62. OM Attributes of a Community-Name Object . . . 998
63. OM Attributes of a Complexity-Limitation Object . . . 998
64. OM Attributes of a Context Object . . . 999
65. OM Attributes of a Create-Object-Instance Object . . . 1001
66. OM Attributes of a Delete-Error Object . . . 1002
67. OM Attributes of a DS-DN Object . . . 1002
68. OM Attributes of a DS-RDN Object . . . 1003
69. OM Attributes of an Entity-Name Object . . . 1003
70. OM Attributes of an Error Object . . . 1004
71. OM Attributes of an Error-Info Object . . . 1004
72. OM Attributes of an Event-Reply Object . . . 1005
73. OM Attributes of an Event-Type-Id Object . . . 1006
74. OM Attributes of an External-AC Object . . . 1006
75. OM Attributes of a Filter-Item Object . . . 1007
76. OM Attributes of a Get-Info-Status Object . . . 1008
77. OM Attributes of an Invalid-Argument-Value Object . . . 1009
78. OM Attributes of a Library-Error Object . . . 1010
79. OM Attributes of a Missing-Attribute-Value Object . . . 1011
80. OM Attributes of a Modification Object . . . 1012
81. OM Attributes of a Multiple-Reply Object . . . 1013
82. OM Attributes of a Network-Address Object . . . 1014
83. OM Attributes of a No-Such-Action Object . . . 1014
84. OM Attributes of a No-Such-Action-Id Object . . . 1015
85. OM Attributes of a No-Such-Argument Object . . . 1015
86. OM Attributes of a No-Such-Event-Id Object . . . 1016
87. OM Attributes of a No-Such-Event-Type Object . . . 1016
88. OM Attributes of an Object-Class Object . . . 1017
89. OM Attributes of an Object-Instance Object . . . 1017
90. OM Attributes of an Object-Syntax Object . . . 1018
91. OM Attributes of a Processing-Failure Object . . . 1018
92. OM Attributes of a Scope Object . . . 1019
93. OM Attributes of a Session Object . . . 1021
94. OM Attributes of a Set-Info-Status Object . . . 1022
95. OM Attributes of a Simple-Syntax Object . . . 1023
96. OM Attributes of an SNMP-Get-Argument Object . . . 1024
97. OM Attributes of an SNMP-Get-Result Object . . . 1024
98. OM Attributes of an SNMP-Response Object . . . 1025
99. OM Attributes of an SNMP-Service-Error Object . . . 1025
100. OM Attributes of an SNMP-Set-Argument Object . . . 1026
101. OM Attributes of an SNMP-Set-Result Object . . . 1027
102. OM Attributes of an SNMP-Trap-Argument Object . . . 1027
103. OM Attributes of a Specific-Error-Info Object . . . 1028
104. OM Attributes of a Substring Object . . . 1029

105. OM Attributes of a Substrings Object . . . 1030
106. OM Attributes of a System-Error Object . . . 1030
107. OM Attributes of a Var-Bind Object . . . 1031
108. Object Identifiers for LNV Attributes . . . 1033
109. Information Syntax for LNV Attribute Value . . . 1033
110. OM Attributes of a CMOT-System-Id . . . 1033
111. Object Identifiers for DMI Object Classes . . . 1034
112. Object Identifiers for DMI Attributes . . . 1034
113. Object Identifiers for DMI Attribute Groups . . . 1037
114. Object Identifiers for DMI Notifications . . . 1037
115. Object Identifiers for DMI Parameters . . . 1038
116. Object Identifiers for DMI Name Bindings . . . 1038
117. Object Identifiers for DMI Packages . . . 1038
118. DMI Attribute Value Syntaxes . . . 1039
119. DMI Notification Information Syntaxes . . . 1041
120. DMI Parameter Value Syntaxes . . . 1042
121. OM Attributes of an Additional-Information . . . 1042
122. OM Attributes of an Alarm-Info . . . 1042
123. OM Attributes of an Alarm-Status . . . 1043
124. OM Attributes of an Allomorphs . . . 1043
125. OM Attributes of an Attribute-Identifier-List . . . 1043
126. OM Attributes of an Attribute-List . . . 1043
127. OM Attributes of a Setof-Attribute-Value-Change-Definition . . . 1043
128. OM Attributes of an Attribute-Value-Change-Definition . . . 1044
129. OM Attributes of an Attribute-Value-Change-Info . . . 1044
130. OM Attributes of an Availability-Status . . . 1044
131. OM Attributes of a Back-Up-Destination-List . . . 1045
132. OM Attributes of a Back-Up-Relationship-Object . . . 1045
133. OM Attributes of a Capacity-Alarm-Threshold . . . 1045
134. OM Attributes of a Control-Status . . . 1045
135. OM Attributes of a Setof-Correlated-Notifications . . . 1045
136. OM Attributes of a Correlated-Notifications . . . 1046
137. OM Attributes of a Correlated-Notifications-1 . . . 1046
138. OM Attributes of a Setof-Counter-Threshold . . . 1046
139. OM Attributes of a Counter-Threshold . . . 1046
140. OM Attributes of a Destination . . . 1047
141. OM Attributes of a Multiple . . . 1047
142. OM Attributes of a Setof-Gauge-Threshold . . . 1047
143. OM Attributes of a Gauge-Threshold . . . 1047
144. OM Attributes of a Group-Objects . . . 1047
145. OM Attributes of a Setof-Intervals-Of-Day . . . 1048
146. OM Attributes of a Intervals-Of-Day . . . 1048
147. OM Attributes of a Management-Extension . . . 1048
148. OM Attributes of a Monitored-Attributes . . . 1048
149. OM Attributes of a Notify-Threshold . . . 1048
150. OM Attributes of a Object-Info . . . 1049
151. OM Attributes of an Observed-Value . . . 1049
152. OM Attributes of a Packages . . . 1049
153. OM Attributes of a Setof-Prioritised-Object . . . 1049
154. OM Attributes of a Prioritised-Object . . . 1050
155. OM Attributes of a Probable-Cause . . . 1050
156. OM Attributes of a Procedural-Status . . . 1050
157. OM Attributes of a Proposed-Repair-Actions . . . 1050
158. OM Attributes of a Relationship-Change-Info . . . 1051

159. OM Attributes of a Security-Alarm-Detector 1051
160. OM Attributes of a Security-Alarm-Info 1051
161. OM Attributes of a Service-User 1052
162. OM Attributes of a Simple-Name-Type 1052
163. OM Attributes of a Specific-Identifier 1052
164. OM Attributes of a Specific-Problems 1052
165. OM Attributes of a State-Change-Info 1053
166. OM Attributes of a Stop-Time 1053
167. OM Attributes of a Setof-Supported-Features 1053
168. OM Attributes of a Supported-Features 1053
169. OM Attributes of a System-Id 1054
170. OM Attributes of a System-Title 1054
171. OM Attributes of a Threshold-Info 1054
172. OM Attributes of a Threshold-Level-Ind 1055
173. OM Attributes of a Up 1055
174. OM Attributes of a Down 1055
175. OM Attributes of a Tide-Mark 1055
176. OM Attributes of a Tide-Mark-Info 1055
177. OM Attributes of a Time24 1056
178. OM Attributes of a Setof-Week-Mask 1056
179. OM Attributes of a Week-Mask 1056

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Trademarks

The following terms, denoted by an asterisk (*) at their first occurrences in this publication, are trademarks of IBM Corporation in the United States or other countries:

AIX
AIXwindows
APPN
IBM
InfoExplorer
NETCENTER
NetView
PS/2
RISC System/6000
SystemView

The following terms, denoted by a double asterisk (**) at their first occurrences in this publication, are trademarks of other companies in the United States or in other countries:

CompuServe	CompuServe, Inc.
Motif	Open Software Foundation
NFS	SUN Microsystems Inc.
X Window System	Massachusetts Institute of Technology

About This Book

The *NetView for AIX Programmer's Reference* provides reference information for programmers who are already familiar with the programming tasks involved in customizing the IBM* NetView* for AIX* program or in writing or customizing network management applications that interface with the NetView for AIX program. This book is to be used in conjunction with the *NetView for AIX Programmer's Guide*.

Who Should Use This Book

Programmers who are customizing the NetView for AIX program or writing or customizing network management applications that are to be used through the NetView for AIX windows interface should refer to this book for detailed descriptions of API functions. These programmers should have programming experience in the following areas:

- C-programming language
- Data communications
- Networking
- AIX Operating System

How to Use This Book

This book is most helpful when the reader understands the following book conventions:

- Highlighting and naming conventions
- Reference page format (including heading definitions)

Highlighting and Operation Naming Conventions

The following highlighting conventions are used in this book, with the noted exceptions:

Bold	Identifies commands and shell script paths (except in reference information), default values, user selections, daemon paths (on first occurrence), and flags (in parameter lists).
Italics	Identifies parameters whose actual names or values are to be supplied by the user, and terms that are defined in the following text.
`Monospace`	Identifies subjects of examples, messages in text, examples of portions of program code, examples of text you might see displayed, information you should actually type, and examples used as teaching aids.

The NetView for AIX operation naming convention used in this book shows the location of the operation in relation to the menu bar or context menu. The naming convention follows the format shown in this example:

```
Monitor..Network Configuration..Addresses
```

In this example, `Monitor` is a menu bar or context menu option, `Network Configuration` is an operation available from the Monitor submenu, and `Addresses` is an option that is available when you select Network Configuration.

Some operations require you to make selections from several layers of submenus before you reach the submenu containing the operation.

Format of Reference Pages

The detailed descriptions of NetView for AIX commands, daemons, files, and applications follow the standard reference page format. Each NetView for AIX reference page may include any of the following sections:

Purpose	Brief description of the major function of the subject
Related Functions	List of functions that are related to and are described in the same reference page as the main function
Syntax	Syntax showing command line options
Dependencies	Description of any dependencies for the use of the subject
Description	Detailed description of the functions and uses of the subject
Parameters	List of parameters associated with a subject and an explanation of the parameter and its possible and default values
Return Values	List of values returned by the subject upon completion or failure
Error Codes	List of error codes returned by the subject upon failure
Flags	List of command line flags associated with the subject, with an explanation of the flag, and its possible and default values
Examples	Specific examples showing command usage and formats of files
Implementation Specifics	Identification of the package of each subject
Libraries	List of libraries to which you need to link to compile a program that uses the function
Files	List of files used by the subject
Warning	Note about a problem that might involve damage to the program
Related Information	List of related subjects in this book, NetView for AIX documentation, Internet Request for Comments, and other information sources

Meaning of Function Numbers

The parenthetical function numbers associated with the heading of each man page have the following meanings:

1 User commands

3 Library routines

4 Files

5 Administrative files

8 Administrative commands

Note: Some of the reference pages refer to reference pages that are not in the *NetView for AIX Programmer's Reference* but that can be accessed through the man command.

Where to Find More Information

The “Bibliography” on page 1107 describes publications that can be helpful when using the NetView for AIX program. The Internet Request for Comments (RFC) documents listed are shipped on the NetView for AIX program installation media and are installed in the /usr/OV/doc directory.

The following sources provide specific information that is not documented in the NetView for AIX Version 4 library:

- The /usr/lpp/nv6000/README file provides additional information about the NetView for AIX program.
- The online help facility provides task, dialog box, and graphical interface information to help you use this program.
- For more information about Simple Network Management Protocol (SNMP), Transmission Control Protocol/Internet Protocol (TCP/IP), and general network basics, the following list contains recommended reading:

  Rose, Marshall T. *The Simple Book: An Introduction to Management of TCP/IP-based Internets*. Englewood Cliffs, NJ: Prentice-Hall, 1994. (ISBN 0-13-177254-6)

  Comer, Douglas. *Internetworking with TCP/IP: Principles, Protocols, and Architecture, Volume 1*. New York, NY: Prentice-Hall, 1991. (ISBN 0-13-468505-9)

  Black, Uyless. *Network Management Standards. The OSI, SNMP, and CMOL Protocols*. New York, NY: McGraw-Hill, 1992. (ISBN 0-07-005554-8)

Chapter 1. Function Tables for NetView for AIX Man Pages

The tables in this section provide the following information about the reference pages in this manual:

- Name of the function or reference page
- Brief description of the purpose
- Page number to see in this book for a complete description.

These tables describe the following functions:

- Graphical User Interface Routines
- SNMP Routines
- XMP Functions
- Filtering and Thresholding Functions
- GTM API Routines
- Miscellaneous Functions
- Introductions
- Collection Facility Routines
- Security Functions
- Client/Server APIs

Graphical User Interface Routines

Table 1 (Page 1 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwAckMapClose(3)	Acknowledges a map close event	528
OVwAckUserSubmapCreate(3)	Acknowledges a user submap create event	530
OVwAddActionCallback(3)	Registers a callback for a registered action	532
OVwAddAlertCallback(3)	Registers handlers of NetView for AIX alerts	535
OVwAddCallback(3)	Registers procedures to process NetView for AIX events	539
OVwAddHelpCallback(3)	Registers a handler for application help requests	541
OVwAddInput(3)	Adds an event source	543
OVwAddMenuItem(3)	Adds a menu item to a menu	545
OVwAddMenuItemFunction(3)	Adds a menu item function to a menu item	547
OVwAddObjMenuItem(3)	Adds an item to the Object Menu	550
OVwAddObjMenuItemFunction(3)	Adds a function to an Object Menu's menu item	552
OVwAddToolPalItem(3)	Adds a tool item to the Tool Window	555
OVwAlertMsg(3)	Issues a NetView for AIX alert message	558
OVwApiIntro(5)	Provides an overview of the OVw API	560
OVwBeginMapSync(3)	Begins map synchronization phase	573
OVwCheckAction(3)	Enables applications to check the validity of other NetView for AIX applications' actions	575

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 2 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwClearBackgroundGraphic(3)	Clears the background picture from the specified submap	801
OVwClearSymbolApp(3)	Clears application interest in a symbol	808
OVwConfirmAcknowledgeObjectsCB(3)[1]	Functions as a callback for an acknowledge object event	578
OVwConfirmAddSymbolCB(3)[1]	Functions as a callback for an add symbol event	833
OVwConfirmAppConfigCB(3)[1]	Functions as a callback for an application configuration change event	839
OVwConfirmCapabililtyChangeCB(3)[1]	Functions as a callback for an object capability change event	580
OVwConfirmCompoundStatusCB(3)[1]	Functions as a callback for compound status events	601
OVwConfirmConnectSymbolsCB(3)[1]	Functions as a callback for a connect symbols event	842
OVwConfirmCreateObjectsCB(3)[1]	Functions as a callback for a create object event	582
OVwConfirmCreateSubmapsCB(3)[1]	Functions as a callback for a create submap event	584
OVwConfirmCreateSymbolsCB(3)[1]	Functions as a callback for a create symbol event	586
OVwConfirmDeleteObjectsCB(3)[1]	Functions as a callback for a delete object event	588
OVwConfirmDeleteSubmapsCB(3)[1]	Functions as a callback for a delete submap event	590
OVwConfirmDeleteSymbolsCB(3)[1]	Functions as a callback for a delete symbol event	846
OVwConfirmDescribeCB(3)[1]	Functions as a callback for a describe change event	849
OVwConfirmExplodeObjectCB(3)[1]	Functions as a callback for an explode object event	592
OVwConfirmHideSymbolsCB(3)[1]	Functions as a callback for a hide symbol event	594
OVwConfirmManageObjectsCB(3)[1]	Functions as a callback for a manage object event	596
OVwConfirmMoveSymbolCB(3)[1]	Functions as a callback for a move symbol event	599
OVwConfirmObjectStatusCB(3)[1]	Functions as a callback for a change object status event	601
OVwConfirmSymbolStatusCB(3)[1]	Functions as a callback for a change symbol status event	601
OVwConfirmUnhideSymbolsCB(3)[1]	Functions as a callback for an unhide symbol event	594

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 3 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwConfirmUnmanageObjectsCB(3)[1]	Functions as a callback for an unmanage object event	596
OVwConfirmUnacknowledgeObjectsCB(3)[1]	Functions as a callback for an unacknowledge object event	578
OVwCopyMapInfo(3)	Allocates memory for OVwMapInfo structure and returns a pointer to a copy of the specified map structure	719
OVwCreateAction(3)	Creates the specified action in the current registration context	603
OVwCreateApp(3)	Creates the specified NetView for AIX application by creating registration information for it	607
OVwCreateComponentSymbol(3)	Creates a symbol representing the object identified by objectId on the child submap of a component object identified by parentId	623
OVwCreateComponentSymbolByName(3)	Creates a symbol representing the object identified by name on the child submap of a component object identified by parentId	623
OVwCreateConnSymbol(3)	Creates a connection symbol representing an object identified by objectId between two icon symbols identified by endpoint1 and endpoint2 on the submap identified by submapId	623
OVwCreateConnSymbolByName(3)	Creates a connection symbol representing an object identified by name between two icon symbols identified by endpoint1 and endpoint2 on a submap of an open map	623
OVwCreateMenu(3)	Creates a menu in the current registration context	611
OVwCreateMenuItem(3)	Creates a menu item in the current registration context	613
OVwCreateObjMenuItem(3)	Creates a menu item in the current registration context	616
OVwCreateSubmap(3)	Creates a submap	619
OVwCreateSymbol(3)	Creates a symbol	623
OVwCreateSymbolByName(3)	Creates a symbol representing the object with the name field value indicated by the specified name	623
OVwCreateSymbolByHostName(3)	Creates a symbol representing the object identified by the specified IP host name	623
OVwCreateSymbolBySelectionName(3)	Creates a symbol representing the object identified by the specified selection name	623
OVwCreateSymbols(3)	Creates symbols	623
OVwDbAppendEnumConstants(3)	Appends constants to an existing enumeration	633
OVwDbCreateField(3)	Creates a field in the object database	635
OVwDbCreateObject(3)	Creates an object in the OVW object database	638

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 4 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwDbCreateObjectByHostname(3)	Creates an object in the OVW object database through a host name	638
OVwDbCreateObjectBySelectionName(3)	Creates an object in the OVW object database through a selection name	638
OVwDbDeleteField(3)	Deletes a field in the object database	635
OVwDbDeleteObject(3)	Deletes an object from the OVW object database	641
OVwDbFieldIdToFieldName	Returns the name of the field that has the field ID fieldId	643
OVwDbFieldNameToFieldId(3)	Returns the field ID of the field that has the field name fieldName	643
OVwDbFreeEnumConstants	Frees the memory allocated for an OVwEnumConstants structure	645
OVwDbFreeFieldBindList	Frees the memory allocated for an OVwFieldBindList structure	654
OVwDbFreeFieldInfo	Frees the memory allocated for an OVwFieldInfo structure	648
OVwDbFreeFieldList	Frees the memory allocated for an OVwFieldList structure	664
OVwDbFreeFieldValue	Frees the memory allocated for an OVwFieldValue structure	650
OVwDbFreeObjectFieldList	Frees the memory allocated for an OVwObjectFieldList structure	656
OVwDbFreeObjectIdList	Frees the memory allocated for an OVwObjectIdList structure	667
OVwDbGetCapabilityFieldValues	Returns a list of all field values for capability fields for a specified object	654
OVwDbGetEnumConstants(3)	Returns a list of all text value sets in the enumerated data type	645
OVwDbGetEnumName	Translates an index into an enumerated constant	645
OVwDbGetEnumValue	Translates an enumerated constant into an index value	645
OVwDbGetFieldBooleanValue	Returns the Boolean value set for a field of type ovwBooleanField for a specified object	650
OVwDbGetFieldEnumByName	Returns the value set for a field of type ovwEnumField for a specified object	650
OVwDbGetFieldEnumByValue	Returns the value set for a field of type ovwEnumField for a specified object	650
OVwDbGetFieldInfo(3)	Returns information about a database field	648
OVwDbGetFieldIntegerValue(3)	Returns the integer value set for a field of type ovwInitField for a specified object	650
OVwDbGetFieldStringValue(3)	Returns the string value set for a field of type ovwStringField for a specified object	650

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 5 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwDbGetFieldValue(3)	Returns the value of a specified field for a specified object	650
OVwDbGetFieldValues(3)	Returns a list of all the field values for a specified object	654
OVwDbGetFieldValuesByObjects(3)	Returns the value set for a specified field for a list of objects	656
OVwDbGetNameFieldValues(3)	Returns a list of all field values for name fields for a specified object	654
OVwGetObjMenuItem(3)	Retrieves registration information for the specified object menu item in the current registration context	616
OVwDbGetUniqObjectName(3)	Returns a unique name for an object	658
OVwDbHostnameToObjectId(3)	Returns the object ID for the object whose IP host name is hostname	660
OVwDbInit(3)	Initializes the OVwDb API	662
OVwDbListFields(3)	Returns a list of object database fields	664
OVwDbListObjectsByFieldValue(3)	Returns a list of objects from the OVW object database that have a single specific value set for a field	667
OVwDbListObjectsByFieldValues(3)	Returns a list of objects from the OVW object database that have all the field values specified by a list of fields	667
OVwDbNameToObjectId(3)	Returns the ObjectID of the object that has a specified name in a specified name field	670
OVwDbObjectIdToHostname(3)	Returns the IP hostname for the object identified by objectId	660
OVwDbObjectIdToSelectionName(3)	Returns the selection name for the object identified by objectId	672
OVwDbSelectionNameToObjectId(3)	Returns the ObjectId of the object that has a selection name matching the value provided by selectionName	672
OVwDbSetEnumConstants(3)	Sets values for a field of type ovwEnumField	674
OVwDbSetFieldBooleanValue(3)	Sets values for a field of type ovwBooleanField	676
OVwDbSetFieldEnumByName(3)	Sets by index the value of a field of type ovwEnumField	676
OVwDbSetFieldEnumByValue(3)	Sets by value the value of a field of type ovwEnumField	676
OVwDbSetFieldIntegerValue(3)	Sets the value for a field of type ovwIntegerField	676
OVwDbSetFieldStringValue(3)	Sets a value for a field of type ovwStringField	676
OVwDbSetFieldValue(3)	Sets the value for the field specified by fieldId for a specified object	676

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 6 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwDbSetHostname(3)	Sets the value of the IP Hostname field for a specified object	679
OVwDbSetSelectionName(3)	Sets the value of the Selection Name field for a specified object	679
OVwDbUnsetFieldValue(3)	Unsets the value of a specified field for a specified object	681
OVwDbUnsetFieldValues(3)	Unsets the values of fields in a list for a specified object	681
OVwDeleteAction(3)	Deletes the specified action in the current registration context	603
OVwDeleteApp(3)	Deletes the specified NetView for AIX application registration	607
OVwDeleteField(3)	Deletes a field in the object database	635
OVwDeleteMenu(3)	Deletes the specified menu from the current registration context	611
OVwDeleteMenuItem(3)	Deletes the specified menu item from the current registration context	613
OVwDeleteObjMenuItem(3)	Deletes registration information for the specified object menu item in the current registration context	616
OVwDeleteSubmap(3)	Deletes a submap from the open map	619
OVwDeleteSymbol(3)	Deletes the symbol identified by symbolId from the open map	623
OVwDeleteSymbols(3)	Deletes symbols in a list from the open map	623
OVwDisplaySubmap(3)	Displays a submap of the open map	683
OVwDoAction(3)	Starts any registered application action	575
OVwDone(3)	Terminates an application's connection to the NetView for AIX program	685
OVwEndMapSync(3)	Ends map synchronization phase	573
OVwEndSessionCB(3)[1]	Functions as a callback for an end-of-session event	686
OVwError(3)	Specifies the error code set by the last OVw API call	688
OVwErrorMsg(3)	Describes OVw API error codes	689
OVwEventIntro(3)	Introduces NetView for AIX graphical user interface events	691
OVwFileDescriptor(3)	Accesses the NetView for AIX program's communications channel	694
OVwFindMenuItem(3)	Finds a menu item	696
OVwFreeActionRegInfo(3)	Frees the memory allocated for an OVwActionRegInfo structure	603
OVwFreeAppRegInfo(3)	Frees the memory allocated for an OVwAppRegInfo structure	607

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 7 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwFreeMapInfo(3)	Frees the memory allocated for an OVwMapInfo structure	719
OVwFreeObjectInfo(3)	Frees the memory allocated for an OVwObjectInfo structure	725
OVwFreeObjectList(3)	Frees the memory allocated for an OVwObjectList structure	745
OVwFreeSubmapInfo(3)	Frees memory allocated for an OVwSubmapInfo structure	733
OVwFreeSubmapList(3)	Frees the memory allocated for an OVwSubmapList structure	747
OVwFreeSymbolInfo(3)	Frees memory allocated for an OVwSymbolInfo structure	735
OVwFreeSymbolList(3)	Frees the memory allocated for an OVwSymbolList structure	750
OVwFreeSymbolTypeList(3)	Frees the memory allocated for an OVwSymbolTypeList structure	753
OVwGetAction(3)	Retrieves registration information for the specified action in the current registration context	603
OVwGetApp(3)	Retrieves registration information for the application that is the current registration context	607
OVwGetAppConfigValues(3)	Gets application configuration parameters	698
OVwGetAppName(3)	Returns the name of the running application	701
OVwGetConnSymbol(3)	Gets a connection symbol	702
OVwGetFirstAction(3)	Returns the name of the first action registered in the current registration context	706
OVwGetFirstMenuItem(3)	Returns the ID of the first menu item registered in the current registration context	708
OVwGetFirstMenuItemFunction(3)	Returns the type and argument for the first function bound to a menu item in the current registration context	710
OVwGetFirstObjMenuItem(3)	Returns the ID of the first object menu item registered in the current registration context	712
OVwGetFirstObjMenuItemFunction(3)	Returns the type and argument for the first function bound to an object menu item in the current registration context	712
OVwGetFirstRegContext(3)	Returns the name of the first application in the NetView for AIX program's list of registered applications	717
OVwGetMapInfo(3)	Returns information about the open map	719
OVwGetMenuItem(3)	Retrieves registration information for the specified menu item in the current registration context	613
OVwGetMenuItemMenu(3)	Returns the ID of the menu to which an item is attached	721

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 8 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwGetMenuItemPath(3)	Returns the path location of a menu item in the NetView for AIX menu bar structure	721
OVwGetMenuPathSeparator(3)	Returns the current string used to separate menu labels in a menu path string	723
OVwGetNextAction(3)	Returns the name of the first action registered in the current registration context	706
OVwGetNextMenuItem(3)	Returns the ID of the next menu item registered in the current registration context	708
OVwGetNextObjMenuItem(3)	Returns the ID of the next object menu item registered in the current registration context	712
OVwGetNextMenuItemFunction(3)	Returns the type and argument for the next function bound to a menu item in the current registration context	710
OVwGetNextObjMenuItemFunction(3)	Returns the type and argument for the next function bound to an object menu item in the current registration context	715
OVwGetNextRegContext(3)	Returns the name of the next application in the NetView for AIX program's list of registered applications	717
OVwGetObjectInfo(3)	Returns map-specific object information	725
OVwGetObjMenuItemMenu(3)	Returns the ID of the object menu to which an item is attached	727
OVwGetObjMenuItemPath(3)	Returns the path location of a menu item in the NetView for AIX menu bar structure	727
OVwGetRegContext(3)	Returns the name of the current registration context	729
OVwGetSelections(3)	Returns the list of selected objects on the open map	731
OVwGetSubmapInfo(3)	Returns information about a submap on the open map	733
OVwGetSymbolInfo(3)	Returns information about a symbol on the open map	735
OVwGetSymbolsByObject(3)	Returns a list of all the symbols that represent an object on the open map	737
OVwHighlightObject(3)	Highlights all the symbols representing a specified object on the open map	739
OVwHighlightObjects(3)	Highlights all the symbols representing the objects in a list on the open map	739
OVwInit(3)	Initializes an application's connection to the NetView for AIX program	741
OVwIsIdEqual(3)	Compares OVw API IDs	743
OVwIsIdNull(3)	Tests an OVw API ID to determine whether it is NULL	743

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 9 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwListObjectsOnMap(3)	Returns a filtered list of the objects on the open map	745
OVwListSubmaps(3)	Returns a filtered list of the submaps on the open map	747
OVwListSymbols(3)	Returns a filtered list of symbols on a submap of the open map	750
OVwListSymbolTypeCaps(3)	Returns a list of the capability field values that would be set if you add an object to the map using this symbol type	753
OVwListSymbolTypes(3)	Returns a list of all the currently registered symbol types	753
OVwLockRegUpdates(3)	Acquires permission for the application to make subsequent calls that modify NetView for AIX registration information	755
OVwMainLoop(3)	Defines a while loop that continuously processes NetView for AIX events and application-registered input events	757
OVwMapCloseCB(3)[1]	Functions as a callback for a map close event	759
OVwMapOpenCB(3)[1]	Functions as a callback for a map open event	761
OVwPeekInputEvent(3)	Determines whether an application's registered input source has input awaiting processing	763
OVwPeekOVwEvent(3)	Determines whether a specific type of NetView for AIX event is awaiting processing	763
OVwPending(3)	Determines whether a NetView for AIX event or application-registered event is awaiting processing	765
OVwProcessEvent(3)	Processes a pending NetView for AIX event	767
OVwQueryAddSymbolCB(3)[1]	Functions as a callback for a query add symbol event	833
OVwQueryAppConfigCB(3)[1]	Functions as a callback for a query application configuration event	839
OVwQueryConnectSymbolsCB(3)[1]	Functions as a callback for a query connect symbols event	842
OVwQueryDeleteSymbolsCB(3)[1]	Functions as a callback for a query delete symbols event	846
OVwQueryDescribeCB[1]	Functions as a callback for a query describe symbols event	849
OVwRegIntro(5)	Introduces NetView for AIX graphical user interface registration files	769
OVwRenameRegContext(3)	Changes the name of a NetView for AIX registration context	795
OVwRemoveActionCallback(3)	Unregisters a callback for a registered action	532

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 10 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwRemoveAlertCallback(3)	Unregisters a callback for a NetView for AIX alert	535
OVwRemoveCallback(3)	Unregisters procedures to process NetView for AIX events	539
OVwRemoveHelpCallback(3)	Unregisters a callback for application help requests	541
OVwRemoveInput(3)	Removes an event source	543
OVwRemoveMenuItem(3)	Removes a menu item from a menu in current context	545
OVwRemoveMenuItemFunction(3)	Removes a menu item function from a menu item in current context	547
OVwRemoveObjMenuItem(3)	Removes a menu item from an object menu in current context	550
OVwRemoveObjMenuItemFunction(3)	Removes a menu item function from an object menu item in current context	552
OVwRemoveToolPalItem(3)	Removes a tool item from the Tool Window	555
OVwSaveRegUpdates(3)	Saves modifications to registration information	797
OVwSelectListChangeCB(3)[1]	Functions as a callback for a selection list change event	799
OVwSetAction(3)	Modifies registration information for the specified action in the current registration context	603
OVwSetApp(3)	Modifies registration information for the application that is the current registration context.	607
OVwSetAppConfigValues(3)	Sets application configuration parameters	698
OVwSetBackgroundGraphic(3)	Sets the background graphic for a specified submap	801
OVwSetMenuItem(3)	Modifies registration information for the specified menu item in the current registration context	613
OVwSetObjMenuItem(3)	Modifies registration information for the specified object menu item in the current registration context	616
OVwSetMenuPathSeparator(3)	Sets the character string used to separate menu labels in a menu path string to a specified value	723
OVwSetRegContext(3)	Sets the name of the current registration context to that of the specified application	729
OVwSetStatusOnObject(3)	Sets the status of all symbols on the open map of the specified object that have the symbol status source ovwObjectStatusSource	803
OVwSetStatusOnObjects(3)	Sets the object status on multiple objects	803

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 11 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwSetStatusOnSymbol(3)	Sets the status of a specified symbol if the symbol has status source ovwSymbolStatusSource and if the application has permission to modify the symbol	803
OVwSetStatusOnSymbols(3)	Sets the symbol status on symbols in a list	803
OVwSetSubmapName(3)	Sets the name of a submap	806
OVwSetSymbolApp(3)	Sets application interest in a symbol	808
OVwSetSymbolBehavior(3)	Sets the behavior of a symbol	810
OVwSetSymbolLabel(3)	Sets the label of a symbol	813
OVwSetSymbolPosition(3)	Sets the position of a symbol	815
OVwSetSymbolStatusSource(3)	Sets the status source of a symbol	820
OVwSetSymbolType(3)	Sets the symbol type of a symbol	822
OVwShowHelp(3)	Requests presentation of help information	825
OVwSubmapCloseCB(3)[1]	Functions as a callback for a submap close event.	827
OVwSubmapOpenCB(3)[1]	Functions as a callback for a submap open event	829
OVwUndoRegUpdates(3)	Destroys all changes to registration information since the last call to OVwSaveRegUpdates or OVwLockRegUpdates	797
OVwUnLockRegUpdates(3)	Releases previously acquired update permissions	755
OVwUserSubmapCreateCB(3)[1]	Functions as a callback for a user submap create event	831
OVwVerifyAdd(3)	Validates the addition of a symbol	833
OVwVerifyAppConfigChange(3)	Validates a change of application configuration values	839
OVwVerifyConnect(3)	Validates the user-selected connect operation for two symbols	842
OVwVerifyDeleteSymbol(3)	Validates the deletion of symbols by a user	846
OVwVerifyDescribeChange(3)	Validates the change of information describing an object by the user	849
OVwXtAddInput(3)	Registers the NetView for AIX event source with X	852
OVwXtAppAddInput(3)	Registers the NetView for AIX event source with X for a specified X application context	852
OVwXtAppMainLoop(3)	Dispatches NetView for AIX events, input events registered with the NetView for AIX program, and X events for a specified X application context	855

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

Table 1 (Page 12 of 12). Graphical User Interface Routines and Their Reference Pages

Routine Name	Description	See Page
OVwXtMainLoop(3)	Dispatches NetView for AIX events, input events registered with the NetView for AIX program, and X events	855

[1]A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.

SNMP Routines

Table 2 (Page 1 of 3). SNMP Routines and Their Reference Pages

Routine Name	Description	See Page
nvSnmpBlockingGetTable(3)	Retrieves an entire table from the MIB in a blocking manner	376
nvSnmpErrString(3)	Returns SNMP-specific error strings	379
nvSnmpGetTable(3)	Retrieves an entire table from the MIB in a non-blocking manner	376
nvSnmpGetTableElement(3)	Retrieves the specified element from the OVsnmpVarBind structure returned by nvSnmpBlockingGetTable or one of its related functions	376
nvSnmpTrapOpenFilter(3)	Opens a session with EMS to receive SNMP filtered traps in a non-X environment	380
nvSnmpXGetTable(3)	Retrieves an entire table from the MIB if XtMainLoop or an equivalent function is used to manage file I/O multiplexing	376
nvSnmpXTrapOpenFilter(3)	Opens a session with EMS to receive SNMP filtered traps in an X environment	380
OVsnmpAddNullVarBind(3)	Creates and initializes a new OVsnmpVarBind data structure	435
OVsnmpAddTypedVarBind(3)	Creates and initializes a new OVsnmpVarBind data structure and allocates space for the value of the variable	435
OVsnmpAddVarBind(3)	Allocates space for and initializes an OVsnmpVarBind data structure for getting and setting variables	435
OVsnmpBlockingSend(3)	Sends an SNMP PDU and receives the response	437
OVsnmpClose(3)	Frees resources allocated by a session created by a call to OVsnmpOpen	440
OVsnmpConfAllocEntry(3)	Allocates dynamic storage for an OVsnmpConfEntry structure	442
OVsnmpConfAllocWcList(3)	Allocates dynamic storage for an OVsnmpConfWcList structure	443
OVsnmpConfClose(3)	Closes an SNMP Configuration Database	444
OVsnmpConfCopyEntry(3)	Allocates a new OVsnmpConfEntry and copies the contents of the old OVsnmpConfEntry to the new one	445

Table 2 (Page 2 of 3). SNMP Routines and Their Reference Pages

Routine Name	Description	See Page
OVsnmpConfCreateEntry(3)	Creates a configuration record in the SNMP Configuration Database	446
OVsnmpConfDbName(3)	Determines the name of the SNMP Configuration Database	448
OVsnmpConfDeleteCache(3)	Removes all cached SNMP configuration data from an open database	449
OVsnmpConfDeleteEntry(3)	Deletes a record from the SNMP Configuration Database	450
OVsnmpConfExportFile(3)	Dumps the contents of the SNMP Configuration Database to a file	452
OVsnmpConfFileName(3)	Determines the pathname of the Version 2 backward-compatibility SNMP configuration file associated with the SNMP Configuration Database	454
OVsnmpConfFreeDest(3)	Frees an OVsnmpConfDest structure and its contents	456
OVsnmpConfFreeEntry(3)	Frees an OVsnmpConfEntry structure and its contents	457
OVsnmpConfFreeWcList(3)	Frees an OVsnmpConfWcList structure and its contents	459
OVsnmpConfOpen(3)	Opens an SNMP Configuration database for subsequent use	460
OVsnmpConfImportFile(3)	Replaces the contents of the SNMP Configuration Database with configuration information obtained from a Version 2 compatible configuration file	463
OVsnmpConfParseEntry(3)	Parses a line in Version 2 ovsnmp.conf file form and produces an OVsnmpConfEntry structure	465
OVsnmpConfPrintCntl(3)	Prints the database control information to stdout	467
OVsnmpConfPrintDest(3)	Prints the resolved SNMP configuration parameters for the target destination to stdout	468
OVsnmpConfPrintEntry(3)	Prints the SNMP configuration parameters for a target, wildcard, or global default to stdout	469
OVsnmpConfReadCntl(3)	Reads the control parameters of the SNMP Configuration Database	470
OVsnmpConfReadDefault(3)	Reads the global default paramenters in the SNMP Configuration Database	472
OVsnmpConfReadEntry(3)	Reads the parameters for the target node from the SNMP Configuration Database	474
OVsnmpConfReadNextDest(3)	Reads the next configuration entry from the SNMP Configuration Database cache	476
OVsnmpConfReadNextEntry(3)	Reads the next configuration entry from the SNMP Configuration Database	478
OVsnmpConfReadWcList(3)	Reads the wildcard entries from the SNMP Configuration Database as a singly linked list	480

Table 2 (Page 3 of 3). SNMP Routines and Their Reference Pages

Routine Name	Description	See Page
OVsnmpConfResolveDest(3)	Returns the resolved SNMP configuration parameters for a target node	482
OVsnmpConfStoreCntl(3)	Stores the control parameters for the SNMP Configuration Database	484
OVsnmpConfStoreDefault(3)	Stores the global default SNMP configuration parameters in the SNMP Configuration Database	486
OVsnmpConfStoreEntry(3)	Stores the SNMP configuration parameters for a target in the SNMP Configuration Database	488
OVsnmpCreatePdu(3)	Allocates an OVsnmpPdu data structure of the specified type	490
OVsnmpDoRetry(3)	Retransmits pending SNMP requests	492
OVsnmpErrString(3)	Returns SNMP-specific error strings	494
OVsnmpFixPdu(3)	Deletes a variable with an error from an SNMP PDU	495
OVsnmpFreePdu(3)	Frees all memory associated with the specified PDU	497
OVsnmpGetRetryInfo(3)	Gets information about pending SNMP requests to be retransmitted	499
OVsnmpIntro(5)	Introduces the ovsnmp library	501
OVsnmpOpen(3)	Establishes an active SNMP session for communication with an SNMP agent	507
OVsnmpRead(3)	Receives SNMP messages on all active sessions	510
OVsnmpRecv(3)	Receives an SNMP PDU for a specified session	512
OVsnmpSend(3)	Sends an SNMP PDU in non-blocking mode	514
OVsnmpTrapOpen(3)	Connects to the trapd daemon and sets up to receive traps in a non-X environment	517
OVsnmpXClose(3)	Frees resources allocated by a session created by a call to OVsnmpXOpen in an X environment	440
OVsnmpXOpen(3)	Establishes an active SNMP session for communication with an SNMP agent in an X environment	507
OVsnmpXSend(3)	Sends an SNMP PDU in non-blocking mode in an X environment	514
OVsnmpXTrapOpen(3)	Connects to the trapd daemon and sets up to receive traps in an X environment	517

WinSNMP Functions

Table 3 (Page 1 of 2). WinSNMP Functions and Their Reference Pages

Routine Name	Description	See Page
SnmpCleanup(3)	Deallocates all WinSNMP application resources	858
SnmpClose(3)	Closes a WinSNMP session	860
SnmpContextToStr(3)	Retrieves a textual context descriptor corresponding to the given WinSNMP context	862
SnmpCountVbl(3)	Counts the number of varbinds in a varbindlist structure	865
SnmpCreatePdu(3)	Creates an SNMP protocol data unit (PDU) for use in subsequent communication requests	867
SnmpCreateSession(3)	Creates a WinSNMP session and initializes resources for subsequent communication functions	869
SnmpCreateVbl(3)	Creates and initializes a new varbindlist structure	872
SnmpDecodeMsg(3)	Decodes the specified SNMP message	874
SnmpDeleteVb(3)	Deallocates resources associated with the specified WinSNMP varbindlist	876
SnmpDuplicatePdu(3)	Duplicates the specified PDU	878
SnmpDuplicateVbl(3)	Duplicates the specified varbindlist structure	880
SnmpEncodeMsg(3)	Encodes an SNMP message without sending it	882
SnmpEntityToStr(3)	Returns a textual string for the given WinSNMP entity	884
SnmpFreeContext(3)	Deallocates resources for the specified WinSNMP context	886
SnmpFreeDescriptor(3)	Deallocates resources associated with the specified WinSNMP descriptor object	888
SnmpFreeEntity(3)	Deallocates resources for the specified WinSNMP entity	890
SnmpFreePdu(3)	Deallocates resources for the specified WinSNMP PDU	892
SnmpFreeVbl(3)	Deallocates resources associated with the specific VBL	894
SnmpGetLastError(3)	Indicates why the last WinSNMP operation failed	896
SnmpGetLastError(3)	Provides a textual description of why the last WinSNMP operation failed	898
SnmpGetPduData(3)	Extracts data from the specified PDU	900
SnmpGetRetransmitMode(3)	Indicates the retransmission mode currently in effect	903
SnmpGetRetry(3)	Retrieves the retry value for the specified entity	905
SnmpGetTimeout(3)	Retrieves timeout information for the specified entity	907

Table 3 (Page 2 of 2). WinSNMP Functions and Their Reference Pages

Routine Name	Description	See Page
SnmpGetTranslateMode(3)	Indicates the entity/context translation mode currently in effect	909
SnmpGetVb(3)	Extracts the varbind identified by the supplied index from a varbindlist structure	911
SnmpOidCompare(3)	Lexicographically compares two object identifiers (OIDs)	913
SnmpOidCopy(3)	Duplicates the specified OID	915
SnmpOidToStr(3)	Converts a WinSNMP OID into a dotted numeric string	917
SnmpRecvMsg(3)	Retrieves results of a completed SNMP request or trap for the specified session	919
SnmpRegister(3)	Registers the calling application to receive or discontinue trap and inform notifications	922
SnmpSelect(3)	Checks the I/O status of multiple file descriptors and message queues, handling SNMP file descriptors transparently	925
SnmpSendMsg(3)	Sends an SNMP message to the specified destination entity	927
SnmpSetPduData(3)	Updates the specified PDU with data supplied by the calling application	930
SnmpSetRetransmitMode(3)	Sets the retransmission mode for subsequent SnmpSendMsg operations	932
SnmpSetRetry(3)	Sets the number of retries for subsequent communication with the specified entity	934
SnmpSetTimeout(3)	Sets the timeout value for the specific entity	936
SnmpSetTranslateMode(3)	Sets the entity/context translate mode	938
SnmpSetVb(3)	Adds and updates varbinds in a varbindlist structure	940
SnmpStartup(3)	Initializes and allocates the necessary resources to perform other WinSNMP functions	942
SnmpStrToContext(3)	Defines a WinSNMP context identified by the input string	945
SnmpStrToEntity(3)	Creates a WinSNMP entity identified by the null-terminated input string	948
SnmpStrToOid(3)	Converts a textual object identifier into an internal WinSNMP OID	951

XMP Functions

Table 4 (Page 1 of 3). XMP Functions and Their Reference Pages

Routine Name	Description	See Page
at_array_to_oid(3)	Encodes an array of integers into an OID	26
at_free(3)	Frees memory that has been allocated by an XMP function call	27

Table 4 (Page 2 of 3). XMP Functions and Their Reference Pages

Routine Name	Description	See Page
at_oid_match(3)	Compares two OM object-identifier values	28
at_oid_to_array(3)	Decodes an OID into an array of integers	29
at_oid_to_str(3)	Decodes an OID into an ASCII string	30
at_str_to_oid(3)	Encodes an ASCII string into an OID	31
mp_abandon(3)	Abandons locally the result of a pending, asynchronous operation or notification	34
mp_action_req(3)	Requests an action from managed objects	36
mp_action_rsp(3)	Replies to a confirmed action request	39
mp_bind(3)	Opens a management session	42
mp_cancel_get_req(3)	Cancels in an orderly manner the result of a pending get operation that is executing in asynchronous mode	44
mp_cancel_get_rsp(3)	Replies to a requested cancel-get operation	47
mp_create_req(3)	Creates a new managed-object instance	49
mp_create_rsp(3)	Replies to a requested create operation	52
mp_delete_req(3)	Deletes managed objects	55
mp_delete_rsp(3)	Replies to a requested delete operation	58
mp_error_message(3)	Returns an error message describing a particular error	61
mp_event_report_req(3)	Reports a notification emitted by a managed object	62
mp_event_report_rsp(3)	Replies to a previously reported management notification	65
mp_get_next_req(3)	Retrieves the next SNMP management information	67
mp_get_req(3)	Retrieves management information	69
mp_get_rsp(3)	Replies to a requested get operation or get-next operation	72
mp_initialize(3)	Initializes the XOM workspace	75
mp_receive(3)	Retrieves the parameter of a management operation or notification	76
mp_set_req(3)	Modifies the attribute values of managed objects	81
mp_set_rsp(3)	Replies to a requested set operation	84
mp_shutdown(3)	Frees or discards a workspace	87
mp_unbind(3)	Terminates a management session	88
mp_version(3)	Negotiates features of the interface and service	90
mp_wait(3)	Suspends the caller until a management message is available from one or more bound sessions	93
om_copy(3)	Duplicates a private object	396

Table 4 (Page 3 of 3). XMP Functions and Their Reference Pages

Routine Name	Description	See Page
om_copy_value(3)	Copies a string value from one private object to another	398
om_create(3)	Creates a new private object	400
om_decode(3)	Creates an unencoded version of an encoded private object	402
om_delete(3)	Deletes a private or service-generated object	404
om_encode(3)	Encodes an OM object	406
om_get(3)	Creates a public copy of all or particular parts of a private object	408
om_instance(3)	Checks the class of an object	412
om_put(3)	Adds or replaces attributes in a private object	414
om_read(3)	Reads a string segment in a private object	417
om_remove(3)	Removes attribute values from a private object	419
om_write(3)	Writes a segment of a string into a private object	421

Filtering and Thresholding Functions

Table 5. Filtering and Thresholding Functions and Their Reference Pages

Routine Name	Description	See Page
nvFilterDefine(3)	Creates a new filtering rule or updates an existing rule	126
nvFilterDelete(3)	Removes a filtering rule from a filter file	129
nvFilterErrorMsg(3)	Retrieves the error message that corresponds to an nvFilter API error return code	130
nvFilterFreeNameList(3)	Frees the memory allocated during creation of the list of filtering rule names	131
nvFilterGet(3)	Retrieves the contents of a filtering rule	132
nvFilterGetNameList(3)	Retrieves a list of all filtering rules in a filter file	134
OVeDeregister(3)	Deregisters the caller from receiving events from the listed network nodes	425
OVeFilterAttr(3)	Builds an event filter attribute that can be used in a call to OVeRegister	427
OVeRegister(3)	Registers the caller with EMS to receive events from the listed network nodes	431

GTM API Routines

Table 6 (Page 1 of 4). GTM API Routines and Their Reference Pages

Routine Name	Description	See Page
nvotChangeArcDetails(3)	Changes the contents of the *details* variable for the specified arc.	137

Table 6 (Page 2 of 4). GTM API Routines and Their Reference Pages

Routine Name	Description	See Page
nvotChangeArcIconInGraph(3)	Changes the icon representing an arc in a graph.	141
nvotChangeArcLabelInGraph(3)	Changes the label on an arc in a graph.	146
nvotChangeArcStatus(3)	Changes one or more status values of an arc.	150
nvotChangeBoxBackground(3)	Changes the background image for the child submap of a box graph.	154
nvotChangeBoxDetails(3)	Changes the contents of the *details* variable for the specified box graph.	157
nvotChangeBoxIconInGraph(3)	Changes the icon representing a box graph in a graph.	160
nvotChangeBoxLabelInGraph(3)	Changes the label on a box in a graph.	163
nvotChangeBoxPositionInGraph(3)	Changes the position of a box graph icon in a graph submap.	166
nvotChangeGraphBackground(3)	Changes the background image for the child submap of a graph.	169
nvotChangeGraphDetails(3)	Changes the contents of the *details* variable for the specified graph.	172
nvotChangeGraphIcon(3)	Changes the icon and label of orphan graphs, boxes, and vertices.	175
nvotChangeGraphIconInGraph(3)	Changes the icon representing a graph in a graph.	178
nvotChangeGraphLabelInGraph(3)	Changes the label on a graph in a graph.	181
nvotChangeGraphPositionInGraph(3)	Changes the position of a graph icon in a graph submap.	184
nvotChangeRootGraphIcon(3)	Changes the icon representing the root graph.	187
nvotChangeRootGraphLabel(3)	Changes the label on the root graph.	190
nvotChangeUnderlyingArcIcon(3)	Changes an underlying arc symbol and label.	193
nvotChangeVertexDetails(3)	Changes the contents of the *details* variable for the specified vertex.	197
nvotChangeVertexIconInBox(3)	Changes the icon representing a vertex in a box graph.	200
nvotChangeVertexIconInGraph(3)	Changes the icon representing a vertex in a graph.	203
nvotChangeVertexLabelInBox(3)	Changes the label on a vertex in a box graph.	206
nvotChangeVertexLabelInGraph(3)	Changes the label on a vertex in a graph.	209
nvotChangeVertexPositionInBox(3)	Changes the position of a vertex icon in a box graph submap.	212
nvotChangeVertexPositionInGraph(3)	Changes the position of a vertex icon in a graph submap.	215
nvotChangeVertexStatus(3)	Changes one or more status values of a vertex.	218
nvotCreateArcInGraph(3)	Creates an arc in a graph.	221
nvotCreateBoxInGraph(3)	Creates a box graph in a graph.	227

Table 6 (Page 3 of 4). GTM API Routines and Their Reference Pages

Routine Name	Description	See Page
nvotCreateGraph(3)	Creates a graph of graph type GRAPH or BOX.	232
nvotCreateGraphInGraph(3)	Creates a graph in a graph.	235
nvotCreateParallelUnderlyingArc(3)	Creates an arc that lies under another arc.	240
nvotCreateProvidingSap(3)	Creates a SAP of type *providing*.	245
nvotCreateRootGraph(3)	Creates a root graph.	249
nvotCreateSerialUnderlyingArc(3)	Creates an arc that lies under another arc.	253
nvotCreateUsingSap(3)	Creates a SAP of type *using*.	258
nvotCreateVertexInBox(3)	Creates a vertex in a box graph.	261
nvotCreateVertexInGraph(3)	Creates a vertex in a graph.	265
nvotDeleteArc(3)	Deletes an arc.	269
nvotDeleteArcFromGraph(3)	Deletes an arc from a graph.	272
nvotDeleteBox(3)	Deletes a box graph.	276
nvotDeleteBoxFromGraph(3)	Deletes a box graph from a graph.	278
nvotDeleteGraph(3)	Deletes a graph.	281
nvotDeleteGraphFromGraph(3)	Deletes a graph from a graph.	283
nvotDeleteProvidingSap(3)	Deletes a SAP of type *providing*.	286
nvotDeleteUnderlyingArc(3)	Deletes an underlying arc relationship to its parent arc.	289
nvotDeleteUsingSap(3)	Deletes a SAP of type *using*.	292
nvotDeleteVertex(3)	Deletes a vertex.	295
nvotDeleteVertexFromBox(3)	Deletes a vertex from a box graph.	297
nvotDeleteVertexFromGraph(3)	Deletes a vertex from a graph.	299
nvotDone(3)	Closes the socket connection to gtmd.	302
nvotFree(3)	Frees memory allocated by a get routine.	304
nvotGetArcsInGraph(3)	Gets a list of all arcs contained in a graph.	307
nvotGetArcObjectId(3)	Gets an arc ObjectId from the OVW database.	310
nvotGetBoxesInGraph(3)	Gets a list of all box graphs contained in a graph.	314
nvotGetBoxObjectId(3)	Gets a box graph ObjectID from the OVW database.	317
nvotGetBoxesWhichVertexIsMemberOf(3)	Gets a list of all boxes of which a vertex is member	320
nvotGetError(3)	Retrieves the error code set by the last function call.	323
nvotGetErrorMsg(3)	Converts a return code into a string.	326
nvotGetGraphObjectId(3)	Gets a graph ObjectId from the OVW database.	327
nvotGetGraphsInGraph(3)	Gets a list of all graphs contained in a graph.	330
nvotGetGraphsWhichArcIsMemberOf(3)	Gets a list of all graphs of which an arc is a member.	333

Table 6 (Page 4 of 4). GTM API Routines and Their Reference Pages

Routine Name	Description	See Page
nvotGetGraphsWhichBoxIsMemberOf(3)	Gets a list of all graphs of which a box is member.	338
nvotGetGraphsWhichGraphIsMemberOf(3)	Gets a list of all graphs of which a child graph is a member.	341
nvotGetGraphsWhichVertexIsMemberOf(3)	Gets a list of all graphs of which a vertex is member	344
nvotGetSapsOnVertex(3)	Gets a list of all SAPs associated with a vertex.	347
nvotGetVertexObjectId(3)	Gets a vertex ObjectId from the OVW database.	350
nvotGetVerticesInBox(3)	Gets a list of all vertices contained in a box graph.	353
nvotGetVerticesInGraph(3)	Gets a list of all vertices contained in a graph.	356
nvotInit(3)	Opens a socket connection to gtmd.	359
nvotSetCenterBoxForGraph(3)	Specifies which box graph icon is to be the center of a star graph submap.	362
nvotSetCenterGraphForGraph(3)	Specifies which graph icon is to be the center of a star graph submap.	365
nvotSetSynchronousCreation(3)	Specifies whether OVw object IDs are to be returned in synchronous mode.	368
nvotVertexHandler(3)	Provides open access to all tables, variables, and operations defined in the NetView for AIX Generic Topology MIB.	370

Collection Facility Routines

Table 7 (Page 1 of 2). Collection Facility Routines and Their Reference Pages

Routine Name	Description	See Page
nvCollectionAdd(3)	Defines new collections of objects	95
nvCollectionAddCallback(3)	Registers procedures to process collection facility events	98
nvCollectionDelete(3)	Deletes a collection definition	100
nvCollectionDone(3)	Closes a connection to the collection facility server	102
nvCollectionError(3)	Returns the error code sent by the last collection facility API call	103
nvCollectionErrorMsg(3)	Returns a textual description of a collection facility API error code	104
nvCollectionEvaluate(3)	Evaluates a rule and returns a list of objects that fit the rule	105
nvCollectionFreeDefn(3)	Frees memory used for collection facility functions	107
nvCollectionGetAllForObject(3)	Obtains a list of all collections the specified object is a member of	109

Table 7 (Page 2 of 2). Collection Facility Routines and Their Reference Pages

Routine Name	Description	See Page
nvCollectionGetInfo(3)	Obtains the description and rule defined for a collection	111
nvCollectionGetTimestamp(3)	Returns the last time a collection was updated	113
nvCollectionIntersect(3)	Finds the intersection of two collections	114
nvCollectionListCollections(3)	Obtains a list of all collections currently defined	116
nvCollectionModify(3)	Modifies a collection definition	117
nvCollectionOpen(3)	Establishes a connection to the collection facility server	119
nvCollectionRead(3)	Reads collection facility events	121
nvCollectionResolve(3)	Obtains a list of all objects currently in a specified collection	122
nvCollectionUnion(3)	Finds the union of two collections	124

Client/Server APIs

Table 8. Client/Server APIs and Their Reference Pages

Routine Name	Description	See Page
NVisClient(3)	Checks to see if an application is running on a client or a server	136
OVDefaultServerName(3)	Determines the name of the default server to which a client should connect	424

Security Functions

Table 9. Security Functions and Their Reference Pages

Routine Name	Description	See Page
nvs_Audit(3)	Defines a format for audit entries to be entered in the security logfile.	384
nvs_DeleteSecContext(3)	Closes a NetView for AIX client's security context with the NetView for AIX security server.	386
nvs_getClientPerms(3)	Obtains a bitmask representation of the permissions a user has for different NetView for AIX functions	388
nvs_isClientAuthorized	Query a user's access to NetView for AIX functions to determine if a user can perform an action	391
nvs_SecErrMsg(3)	Returns status message from security API calls	394
nvs_isSecOn(3)	Determines whether NetView for AIX security is active	395

Miscellaneous Functions

Table 10. Miscellaneous Functions and Their Reference Pages

Routine Name	Description	See Page
gtmdump(3)	Monitors GTM trap reception and dumps the contents of the GTM database to a file for problem determination purposes.	32
OVmib_get_objid_name(3)	Converts a MIB variable object identifier to its textual name	433
OVmib_read_objid(3)	Converts a MIB variable name to its object identifier format	434
OVsDone(3)	Notifies ovspmd that a well-behaved object manager is exiting	520
OVsInit(3)	Returns a file descriptor for interprocess communication with ovspmd	520
OVsInitComplete(3)	Notifies ovspmd when a well-behaved object manager has finished initializing	520
OVsPMD_API(3)	Describes routines for well-behaved daemon process in the NetView for AIX program	520
OVsReceive(3)	Receives a command from ovspmd when called by a well-behaved object manager	520
OVuLog(3)	Enables programs to issue logging messages through the nettl logging facility	522
OVuTL(3)	Enables programs to provide logging and tracing output	522
OVwTLInit(3)	Initializes the software and hostname fields in the logging and tracing output	522
OVwTrace(3)	Enables programs to issue tracing messages through the nettl tracing facility	522
XnvApplicationShell(3)	Functions as the main top-level window for an application managed by the NetView for AIX graphical user interface	953
XnvTopLevelShell(3)	Functions as the main top-level window for an application managed by the NetView for AIX graphical user interface	956

Introductions

Table 11. API Introductions and Their Reference Pages

Page Name	Description	See Page
OVsnmpIntro(5)	Introduces the SNMP API	501
OVwApiIntro(5)	Introduces the OVw (End User Interface) API	560
OVwRegIntro(5)	Introduces the registration files used by NetView for AIX	769

Chapter 2. Reference Pages

This chapter contains the reference (man) pages for the NetView for AIX program. These reference pages are organized alphabetically. You can also access these reference pages through the `Help..NetView for AIX Library` menu option or by using the man command.

at_array_to_oid(3)

Purpose

Encodes an array of integers into an OID

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_return_code at_array_to_oid ( unsigned int num_element,
                                 unsigned int *obj_id_array,
                                 OM_string *new_obj_id );
```

Description

The at_array_to_oid creates an Object Identifier string from an array of integers. The XOM API requires that Object Identifier strings be input to the API as BER encoded Object Identifiers. The BER encoded string is returned in malloc'ed memory, which you should free by calling at_free.

Parameters

num_element Specifies the number of elements in the array of integers.

obj_id_array Specifies a pointer to the first integer in the array of integers to be encoded.

new_obj_id Specifies a pointer to the newly encoded Object Identifier string. When the string is no longer needed, the memory can be deallocated with the at_free command.

Return Values

If successful, at_array_to_oid returns a value of [OM_SUCCESS]. If unsuccessful, at_array_to_oid returns one of the following error codes.

Error Codes

[OM_MEMORY_INSUFFICIENT] The service cannot allocate the main memory it needs to complete the function.

[OM_POINTER_INVALID] In the C-language interface, a pointer that is not valid was provided as a function argument or as the receptacle for a function result.

[OM_WRONG_VALUE_LENGTH] An attribute has, or would have, a value that violates the value length constraints in force.

[OM_WRONG_VALUE_MAKEUP] An attribute has, or would have, a value that violates a constraint of the value's syntax.

at_free(3)

Purpose

Frees memory that has been allocated by an XMP function call

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_return_code at_free(void *ptr);
```

Parameters

ptr Specifies a pointer to the memory that needs to be freed

Return Values

The at_free command always returns [OM_SUCCESS].

at_oid_match(3)

Purpose

Compares two OM object identifier values

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_boolean at_oid_match( OM_object_identifier *oid1,
                         OM_object_identifier *oid2 );
```

Parameters

oid1 Specifies the first object identifier string

oid2 Specifies the second object identifier string

Return Values

If the strings are equal, at_oid_match returns the constant [OM_TRUE]. If the strings are unequal, it returns the constant [OM_FALSE].

at_oid_to_array(3)

Purpose

Decodes an OID into an array of integers

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_return_code at_oid_to_array ( OM_string obj_id,
                                 unsigned int *num_element,
                                 unsigned int **obj_id_array );
```

Description

The at_oid_to_array command creates an array of integers from a BER encoded Object Identifier string. The XOM API returns Object Identifier strings in a BER encoded format. This routine decodes the BER and places the data into an array of integers allocated by the function. When no longer needed, you should free the array of integers by calling at_free.

Parameters

obj_id	Specifies the BER encoded Object Identifier string
num_element	Specifies the number of elements of the array is returned in this integer
obj_id_array	Specifies a pointer in which the address of the newly created array will be placed. The array can be freed by using at_free when no longer needed.

Return Values

If successful, at_oid_to_array returns a value of [OM_SUCCESS]. If unsuccessful, at_oid_to_array returns one of the following error codes.

Error Codes

[OM_POINTER_INVALID]	In the C-language interface, a pointer that is not valid was specified as a function argument or as the receptacle for a function result.
[OM_WRONG_VALUE_LENGTH]	An attribute has, or would have, a value that violates the value length constraints in effect.

at_oid_to_str(3)

Purpose

Decodes an OID into an ASCII string

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_return_code at_oid_to_str ( OM_string obj_id,
                               char **obj_id_str );
```

Description

The at_oid_to_str command creates an ASCII string from a BER encoded Object Identifier string. The XOM API returns Object Identifier strings in a BER encoded format. This routine decodes the BER and places the data into an ASCII string of integers separated by periods (for example, 1.3.6.1.2.1.1.2.0). The memory for the ASCII string is allocated by the interface and should be freed after it is no longer needed by calling at_free.

Parameters

obj_id Specifies the BER encoded Object Identifier string

obj_id_str Specifies a pointer to the ASCII representation of the object identifier string that is returned in this pointer. You can free this string by calling at_free.

Return Values

If successful, at_oid_to_str command returns a value of [OM_SUCCESS]. If unsuccessful, at_oid_to_str command returns one of the following error codes.

Error Codes

[OM_MEMORY_INSUFFICIENT]
The service cannot allocate the main memory it needs to complete the function.

[OM_WRONG_VALUE_LENGTH]
An attribute has, or would have, a value that violates the value length constraints in effect.

[OM_WRONG_VALUE_MAKEUP]
An attribute has, or would have, a value that violated a value syntax constraint in effect.

at_str_to_oid(3)

Purpose

Encodes an ASCII string into an OID

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_return_code at_str_to_oid ( char *obj_id_str,
                               OM_string *new_obj_id );
```

Description

The at_str_to_oid command creates an Object Identifier string from a character string. The character string is a sequence of integers represented in ASCII characters separated by periods, (for example, 1.3.6.1.2.1.1.2.0). The XOM API requires that Object Identifier strings be input to the API as BER encoded Object Identifiers. The BER encoded string is returned in malloc'ed memory, which should be freed by calling at_free.

Parameters

obj_id_str — Specifies a pointer to a string of characters which represent the Object Identifier

new_obj_id — Specifies a pointer to the newly encoded Object Identifier string. When the string is no longer needed, the memory can be deallocated with the *at_free* function.

Return Values

If successful, at_str_to_oid returns a value of [OM_SUCCESS]. If unsuccessful, at_str_to_oid returns one of the following error codes.

Error Codes

[OM_MEMORY_INSUFFICIENT] — The service cannot allocate the main memory it needs to complete the function.

[OM_POINTER_INVALID] — In the C-language interface, a pointer that is not valid was specified as a function argument or as the receptacle for a function result.

[OM_WRONG_VALUE_LENGTH] — An attribute has, or would have, a value that violates the value length constraints in effect.

[OM_WRONG_VALUE_MAKEUP] — An attribute has, or would have, a value that violated a value syntax constraint in effect.

gtmdump(8)

Purpose

Displays the contents of gtmd database

Syntax

gtmdump [-h] [-g *filename***] [-o** *filename***] [-d** *filename***]**

Description

The gtmdump command is a tool for troubleshooting problems with the GTM database. It provides two functions:

- Provides a complete view of the GTM database
- Monitors GTM trap processing

Flags

-h Displays the usage screen.

-g [filename] Causes gtmdump to issue generic GET operations to all Open Topology MIB tables. Each table entry is printed to the output file. If a *filename* in not specified, the output is printed to stdout.

The tables are dumped in the following sequence:

- Vertex
- SimpleConnection
- UnderlyingConnection
- Arc
- UnderlyingArc
- Graph
- Member
- MemberArc
- AttachedArc
- Additional Graph information
- Additional Member information
- Sap

-o filename Monitor mode. Each processed notification is appended to a file, in a table entry format, and in the same sequence as they are processed by gtmd. A *filename* is required if you specify this option.

-d filename Monitor mode. The function is similar to option [-o], except that the output is in a format that assists a NetView for AIX support person in analyzing a problem. A *filename* is required if you specify this option.

Pressing *Enter* terminates the options [-o] and [-d].

Examples

The following sequence of steps sets up the gtmdump tool to monitor gtmd output. The notifications are printed in the file 'dump_D.out', which can be sent to NetView for AIX support to create the problem again.

1. Clear databases
2. Issue **ovstart** to start daemons
3. gtmdump -d dump_D.out
4. Start discovery/manager process
5. End gtmdump tool

Related Information

- See ovobjprint.

mp_abandon(3)

Purpose

Abandons locally the result of a pending, asynchronous operation or notification

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_abandon( OM_private_object session,
                      OM_sint invoke_id );
```

Description

- This function abandons the result of an outstanding, asynchronous function call. The function is no longer outstanding after this function returns, and the result (or the remaining results, in the case of multiple linked replies) is never returned by mp_receive.

  Note: The abandon function call does not abandon the outstanding asynchronous call itself, but only its results.

- The mp_abandon function call is successful even if the operation or notification to be abandoned no longer exists or is not confirmed. In this case, the abandon operation is without effect.

Parameters

session

Specifies the management session in which the confirmed operation or notification was requested. This is the private object of the OM class Session that was previously returned from mp_bind.

invoke_id

Specifies the specific outstanding asynchronous operation submitted through the *session* to be terminated. If the outstanding operation is a nonconfirmed service, the abandon operation is without effect. The value of *invoke_id* must be the value returned by the function call that initiated the asynchronous management operation to be abandoned.

Return Values

If successful, mp_abandon returns the constant [MP_SUCCESS]. If unsuccessful, mp_abandon returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

  [BAD_CLASS] An argument that was not valid was specified.

  [BAD_PROCEDURAL_USE] A linked reply that was not valid was specified.

  [BAD_SESSION] A session that was not valid was specified.

  [MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

  [SESSION_TERMINATED] The session is terminated and the results of an outstanding operation is no longer available.

Related Information

- See “mp_bind(3)” on page 42.

mp_action_req(3)

Purpose

Requests an action from managed objects

Syntax

```
#include <xom.h>
#include <xmp.h>
MP_status mp_action_req( OM_private_object session,
                         OM_private_object context,
                         OM_object argument,
                         OM_private_object *result_return,
                         OM_sint *invoke_id_return );
```

Description

This function can be requested in a confirmed mode or a nonconfirmed mode. A reply is expected in confirmed mode, while none is expected in nonconfirmed mode.

This function can be called in both synchronous and asynchronous modes.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the information supplied as the parameter of an action to be performed. This information is an instance of a subclass of the OM class Action-Argument. For a CMIS action operation, the supplied parameter is an instance of the OM class CMIS-Action-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, when the operation was requested in a confirmed mode, the result can be one of the following:

- An instance of the subclass of the OM class Action-Result. For a CMIS action request, the result is an instance of the OM class CMIS-Action-Result.
- An instance of the OM class Multiple-Reply, which is a set of instances of the subclass of the OM class Linked-Reply-Argument. Each instance of the OM class CMIS-Linked-Reply-Argument contains one of the following OM attributes:

  [ACTION_ERROR] A partial, negative result of a confirmed action operation.

  [ACTION_RESULT] A partial, successful result of a confirmed action operation.

[PROCESSING_FAILURE]
When processing the operation, a general failure was encountered after the partial results were sent.

Otherwise, in nonconfirmed mode, no results are expected and the *result_return* parameter is undefined. The constant [MP_ABSENT_OBJECT] indicates the absence of a result.

In asynchronous mode, this parameter is undefined.

invoke_id_return Specifies the invoke identification of the initiated management operation, when invoked in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined. It is applicable only in a confirmed mode.

Return Values

If used in asynchronous mode, the return value for mp_action_req indicates whether the action was completed or whether it was initiated. If used in asynchronous mode, it indicates whether the action was initiated.

If successfully completed when used in synchronous mode or successfully initiated when used in asynchronous mode, mp_action_req returns the constant [MP_SUCCESS]. If uncompleted when used in synchronous mode or not initiated when used in asynchronous mode, mp_action_req returns an error code.

Error Codes

If unsuccessful, mp_action_req returns one of the following error codes.

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any problem value for a Communications-Error
- Any problem value for a System-Error
- One of the following problem values for a CMIS-Service-Error:

  [ACCESS_DENIED] The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT]
  The managed-object instance is not a member of the specified class.

  [COMPLEXITY_LIMITATION] The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

  [INVALID_ARGUMENT_VALUE]
  The event argument value was out of range or otherwise inappropriate.

  [INVALID_FILTER] Contains an assertion that is not valid or an unrecognized logical operator.

  [INVALID_SCOPE] The scope value is not valid.

  [NO_SUCH_ACTION] The action type is not recognized.

  [NO_SUCH_ARGUMENT] The event or action is not recognized.

  [NO_SUCH_OBJECT_CLASS] The managed-object class is not recognized.

  [NO_SUCH_OBJECT_INSTANCE]
  The managed-object instance is not recognized.

[PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

[SYNCHRONIZATION_NOT_SUPPORTED]
This type of synchronization is not supported.

- One of the following problem values for a Library-Error:

[BAD_ARGUMENT] An argument that was not valid was specified.

[BAD_CLASS] The OM class of either an argument, result, linked-reply, or error is not supported for this option.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

[SESSION_TERMINATED]
The session is terminated and the results of an outstanding operation is no longer available.

[SIZE_LIMIT_EXCEEDED]
The maximum number of linked responses, about which the requested service should return information, has been reached.

[TIME_LIMIT_EXCEEDED]
The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). This OM attribute can be absent.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_action_rsp(3)” on page 39.
- See “mp_bind(3)” on page 42.
- See “mp_receive(3)” on page 76.

mp_action_rsp(3)

Purpose

Replies to a confirmed action request

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_action_rsp( OM_private_object session,
                         OM_private_object context,
                         OM_object response,
                         OM_sint invoke_id );
```

Parameters

session

Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context

Specifies the management context to be used for this operation. This parameter must be a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT].

response

Specifies the information supplied as the response to an action. This information can be one of the following:

- An instance of a subclass of the OM class Linked-Reply-Argument. An instance of the OM class CMIS-Linked-Reply-Argument expresses a partial result (linked reply). This instance contains one of the following OM attributes:

  [ACTION_ERROR] A partial, negative result of a confirmed action operation.

  [ACTION_RESULT] A partial, successful result of a confirmed action operation.

  [PROCESSING_FAILURE]
  When processing the operation, a general failure was encountered after the partial results were sent.

- An instance of a subclass of the OM class Action-Result, which is the information supplied as the single reply to an action. This reply indicates the successful completion of the operation. For a CMIS operation, the response is an instance of the OM class CMIS-Action-Result.

- An instance of a subclass of the OM class Service-Error, which indicates the failure of the action. For a CMIS operation, one of the following problem values for a CMIS-Service-Error, as well as its associated parameter, can be sent as a reply:

  [ACCESS_DENIED] The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT]
  The managed-object instance is not a member of the specified class.

[COMPLEXITY_LIMITATION] The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

[INVALID_ARGUMENT_VALUE] The event argument value was out of range or otherwise inappropriate.

[INVALID_FILTER] Contains an assertion that is not valid or an unrecognized logical operator.

[INVALID_SCOPE] The scope value is not valid.

[NO_SUCH_ACTION] The action type is not recognized.

[NO_SUCH_ARGUMENT] The event or action is not recognized.

[NO_SUCH_OBJECT_CLASS] The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE] The managed-object instance is not recognized.

[PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

[SYNCHRONIZATION_NOT_SUPPORTED] This type of synchronization is not supported.

- The constant [MP_ABSENT_OBJECT], which indicates one of the following possibilities:
  - The result is NULL (absent) because no object was selected.
  - This is the last response following a chain of linked replies.

invoke_id

Specifies the invoke identification of the requested operation to which the reply applies.

Return Values

If successful, mp_action_rsp returns the constant [MP_SUCCESS.] If unsuccessful, mp_action_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that is not valid was specified.

[BAD_ERROR] A service error that is not valid was specified.

[BAD_LINKED_REPLY] A linked reply that is not valid was specified.

[BAD_RESULT] A result that is not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION]
The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use in this session.

[SESSION_TERMINATED]
The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_action_req(3)” on page 36.
- See “mp_bind(3)” on page 42.
- See “mp_receive(3)” on page 76.

mp_bind(3)

Purpose

Opens a management session

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_bind( OM_object session,
                   OM_workspace workspace,
                   OM_private_object *bound_session_return );
```

Parameters

session

Specifies a manager session, together with other details of the service required. This parameter can be either a public object or a private object. The constant [MP_DEFAULT_SESSION] can also be used as the value of this parameter, causing a new session to be created with default values for all its OM attributes. If the OM attribute *requestor-Title* is specified, only one session can be opened with the same value of this OM attribute.

The OM attribute *requestor-Title* is required for manager sessions that need to process indications using the mp_receive function call. This is normal for agents, and for managers that expect to process incoming event reports.

workspace

Specifies the workspace (obtained from a call to mp_initialize), which is to be associated with the session. All function results from management operations using this session are returned as private objects in this workspace. If the *session* parameter is a private object, it must be a private object in this workspace.

bound_session_return

Specifies a private object. Upon successful completion, it contains an instance of a management session, which can be used as a parameter to other functions (for example, mp_get_req). This parameter is a new private object if the value of *session* was MP_DEFAULT_SESSION or a public object; otherwise, it is the private object that was supplied as a parameter. If a private object was supplied, the *session* provided should not be already in use. The function supplies default values for any of the OM attributes that were not present in the session instance supplied as a parameter. This function also sets the value of the *file-Descriptor* OM attribute.

Return Values

If successful, mp_bind returns the constant [MP_SUCCESS]. If unsuccessful, mp_bind returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any problem value for a Communications-Error
- Any problem value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_ADDRESS]	An address that is not valid was specified.
[BAD_SESSION]	A session that was not valid was specified.
[BAD_TITLE]	A title that was not valid was specified.
[MISCELLANEOUS]	A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.
[NOT_SUPPORTED]	An attempt was made to use an option that is not available or was not agreed upon for use in this session.
[TOO_MANY_SESSIONS]	Additional management sessions can not be started.

Related Information

- See “mp_unbind(3)” on page 88.

mp_cancel_get_req(3)

Purpose

Cancels in an orderly manner the result of a pending get operation that is executing in asynchronous mode

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_cancel_get_req( OM_private_object session,
                             OM_private_object context,
                             OM_object argument,
                             OM_sint *invoke_id_return );
```

Description

- This function cancels the result of an outstanding, asynchronous mp_get_req function call in an orderly manner. The mp_get_req function is no longer outstanding after the service confirmation of the cancel-get operation is received by mp_receive. Any subsequent replies to the get operation are not returned by mp_receive.
- This service is defined as a confirmed service. A single reply is expected.
- This function can be called in both synchronous and asynchronous modes.

Parameters

session Specifies the management session in which the get operation was requested. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the specific outstanding asynchronous get operation to be terminated. The supplied parameter is an instance of a subclass of the OM class Cancel-Get-Argument. For an instance of the OM class CMIS-Cancel-Get-Argument, the value of the OM attribute *get-Invoke-Id* must be the value returned by the mp_get_req function call that initiated the get operation to be canceled.

invoke_id_return Specifies the invoke identification of the initiated cancel-get asynchronous operation. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined.

Return Values

The return value for this function indicates whether mp_cancel_get_req was initiated.

If initiated and successfully completed when used in synchronous mode or asynchronous mode, mp_cancel_get_req returns the constant [MP_SUCCESS.]

If the action did not complete successfully when used in synchronous mode, mp_cancel_get_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any problem value for a Communications-Error
- Any problem value for a System-Error
- One of the following *problem* values for a CMIS-Service-Error:

  [MISTYPED_OPERATION]
  : The invoke identifier of the get operation does not refer to a get operation.

  [NO_SUCH_INVOKE_ID]
  : The invoke identifier of the get operation is not recognized.

  [PROCESSING_FAILURE]
  : A general failure was encountered during processing of the operation.

- One of the following problem values for a Library-Error:

  [BAD_CONTEXT]
  : A context argument that was not valid was specified.

  [BAD_SESSION]
  : A session that was not valid was specified.

  [MISCELLANEOUS]
  : A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

  [NO_SUCH_OPERATION]
  : The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

  [NOT_SUPPORTED]
  : An attempt was made to use an option that is not available or was not agreed upon for use in this session.

  [SESSION_TERMINATED]
  : The session was terminated and the results of an outstanding operation are no longer available.

If the action did not complete successfully when used in asynchronous mode, mp_cancel_get_req returns one of the following error codes:

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- One of the following *problem* values for a Library-Error:

[BAD_CONTEXT]	A context argument that was not valid was specified.
[BAD_SESSION]	A session that is not valid was specified.
[MISCELLANEOUS]	A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.
[NO_SUCH_OPERATION]	The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.
[NOT_SUPPORTED]	An attempt was made to use an option that is not available or was not agreed upon for use in this session.
[SESSION_TERMINATED]	The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_cancel_get_rsp(3)” on page 47.
- See “mp_receive(3)” on page 76.

mp_cancel_get_rsp(3)

Purpose

Replies to a requested cancel-get operation

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_cancel_get_rsp( OM_private_object session,
                             OM_private_object context,
                             OM_object response,
                             OM_sint invoke_id );
```

Description

- A last reply to the canceled, outstanding get operation has to be issued to indicate the completion of the get operation. That last reply contains the service error *canceled-operation*.
- No further replies to the canceled operation are issued.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT].

response Specifies the information supplied as the response to a cancel-get operation. This information can be one of the following:

- The constant [MP_ABSENT_OBJECT], which indicates the successful completion of the operation.
- An instance of a subclass of the OM class Service-Error, which indicates the failure of the operation.

  For an instance of the OM class CMIS-Service-Error, one of the following *problem* values for a CMIS-Service-Error, as well as its associated *parameter*, can be sent as a reply:

  [MISTYPED_OPERATION]
  The invoke identifier of the get operation does not refer to a get operation.

  [NO_SUCH_INVOKE_ID] The invoke identifier of the get operation is not recognized.

  [PROCESSING_FAILURE]
  A general failure was encountered during processing of the operation.

invoke_id Specifies the invoke identification of the requested cancel-get operation to which the reply applies.

Return Values

If successful, mp_cancel_get_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_cancel_get_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any problem value for a Communications-Error
- Any problem value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_ERROR] A service error that is not valid was specified.

[BAD_PROCEDURAL_USE] The procedure used for linked replies does not comply with the ISO and X/Open standards, or the permitted service primitive chaining was violated.

[BAD_SESSION] A session that is not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION] The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[SESSION_TERMINATED] The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_cancel_get_req(3)” on page 44.
- See “mp_receive(3)” on page 76.

mp_create_req(3)

Purpose

Creates a new managed-object instance

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_create_req( OM_private_object session,
                         OM_private_object context,
                         OM_object argument,
                         OM_private_object *result_return,
                         OM_sint *invoke_id_return );
```

Description

- This function is used to request the creation of a new managed-object instance.
- This function is defined as a confirmed service. A single reply is expected.
- This function can be called in both synchronous and asynchronous modes.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the information supplied as the parameter of a create operation. This information is an instance of a subclass of the OM class Create-Argument. For a CMIS create operation, the supplied parameter is an instance of the OM class CMIS-Create-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, the result may be one of the following:

- An instance of a subclass of the OM class Create-Result. For a CMIS operation, the result is an instance of the OM class CMIS-Create-Result.
- The constant [MP_ABSENT_OBJECT], which may be returned if the requester of the create operation provided the name of the new managed-object instance.

invoke_id_return Specifies the invoke identification of the initiated management operation, when invoked in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined.

Return Values

The return value for this function indicates whether the create operation was completed, if used in synchronous mode, or whether the create operation was initiated, if used in asynchronous mode.

If successful, mp_create_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_create_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any problem value for a Communications-Error
- Any problem value for a System-Error
- One of the following problem values for a CMIS-Service-Error:

  [ACCESS_DENIED]
  : The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT]
  : The managed-object instance is not a member of the specified class.

  [DUPLICATE_MANAGED_OBJECT_INSTANCE]
  : The new managed-object instance value provided by the invoker of the create operation is already registered for a managed object of the specified class.

  [INVALID_ATTRIBUTE_VALUE]
  : The attribute value was out of the valid range or otherwise inappropriate.

  [INVALID_OBJECT_INSTANCE]
  : The name of the object instance does not comply with the naming rules.

  [MISSING_ATTRIBUTE_VALUE]
  : A required attribute value was not specified, and a default value was not available.

  [NO_SUCH_ATTRIBUTE]
  : The identifier of an attribute, or an attribute group, is not recognized.

  [NO_SUCH_OBJECT_CLASS]
  : The managed-object class is not recognized.

  [NO_SUCH_OBJECT_INSTANCE]
  : The managed-object instance is not recognized.

  [NO_SUCH_REFERENCE_OBJECT]
  : The reference-object instance is not recognized.

  [PROCESSING_FAILURE]
  : A general failure was encountered during the processing of an operation.

- One of the following problem values for a Library-Error:

  [BAD_ARGUMENT] An argument that is not valid was specified.

  [BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

  [BAD_CONTEXT] A context argument that was not valid was specified.

  [BAD_SESSION] A session that is not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use in this session.

[SESSION_TERMINATED]
The session was terminated and the results of an outstanding operation are no longer available.

[TIME_LIMIT_EXCEEDED]
The maximum elapsed time within which the requested service must be provided has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. Its syntax is object(Multiple-Reply). This OM attribute can be absent.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_bind(3)” on page 42.
- See “mp_cancel_get_rsp(3)” on page 47.
- See “mp_receive(3)” on page 76.

mp_create_rsp(3)

Purpose

Replies to a requested create operation

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_create_rsp( OM_private_object session,
                         OM_private_object context,
                         OM_object response,
                         OM_sint invoke_id );
```

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT].

response Specifies the information supplied as the single response to a create operation. This information can be one of the following:

- An instance of a subclass of the OM class Create-Result, which indicates the successful completion of the operation. For a CMIS create operation, the response is an instance of the OM class CMIS-Create-Result.
- The constant [MP_ABSENT_OBJECT], which is returned if the create operation was successful and if the requester of the create operation provided the name of the new managed-object instance.
- An instance of a subclass of the OM class Service-Error, which indicates the failure of the requested operation. For a CMIS create operation, one of the following *problem* values for a CMIS-Service-Error, as well as its associated *parameter*, can be sent as a reply:

  [ACCESS_DENIED]
  The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT]
  The managed-object instance is not a member of the specified class.

  [DUPLICATE_MANAGED_OBJECT_INSTANCE]
  The new managed-object instance value provided by the invoker of the create operation is already registered for a managed object of the specified class.

  [INVALID_ATTRIBUTE_VALUE]
  The attribute value was out of the valid range or otherwise inappropriate.

  [INVALID_OBJECT_INSTANCE]
  The name of the object instance does not comply with the naming rules.

  [MISSING_ATTRIBUTE_VALUE]
  A required attribute value was not specified, and a default value was not available.

[NO_SUCH_ATTRIBUTE]
The identifier of an attribute, or an attribute group, is not recognized.

[NO_SUCH_OBJECT_CLASS]
The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[NO_SUCH_REFERENCE_OBJECT]
The reference-object instance is not recognized.

[PROCESSING_FAILURE]
A general failure was encountered during the processing of an operation.

invoke_id Specifies the invoke identification of the requested operation to which the reply applies.

Return Values

The mp_create_rsp command returns a value to indicate whether the response was completed.

If successful, mp_create_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_create_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a System-Error
- Any *problem* value for a Communications-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_ERROR] A service error that is not valid was specified.

[BAD_RESULT] A result that is not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION]
The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[SESSION_TERMINATED]
The session is terminated and the results of an outstanding operation is no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_create_req(3)” on page 49.
- See “mp_receive(3)” on page 76.

mp_delete_req(3)

Purpose

Deletes managed objects

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_delete_req( OM_private_object session,
                         OM_private_object context,
                         OM_object argument,
                         OM_private_object *result_return,
                         OM_sint *invoke_id_return );
```

Description

This service is defined as a confirmed service, which can be called in both synchronous and asynchronous modes. A reply is expected.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the information supplied as the parameter of a delete operation. This information is an instance of a subclass of the OM class Delete-Argument. For a CMIS delete operation, the supplied parameter is an instance of the OM class CMIS-Delete-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, the result can be one of the following:

- An instance of a subclass of the OM class Delete-Result. For a CMIS operation, the result is an instance of the OM class CMIS-Delete-Result.
- An instance of the OM class Multiple-Reply, which is a set of instances of a subclass of the OM class Linked-Reply-Argument. Each instance of the OM class CMIS-Linked-Reply-Argument contains one of the following OM attributes:
  - *delete-Error*
  - *delete-Result*
  - *processing-Failure*
- The constant [MP_ABSENT_OBJECT], which indicates the absence of a result, if no managed object was selected for the operation.

invoke_id_return Specifies the invoke identification of the initiated management operation when invoked in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined.

Return Values

The return value for this function indicates whether the delete operation was completed if used in synchronous mode; or whether the delete operation was initiated, if used in asynchronous mode.

If successful, mp_delete_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_delete_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a CMIS-Service-Error:

  [ACCESS_DENIED] The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT] The managed-object instance is not a member of the specified class.

  [COMPLEXITY_LIMITATION] The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

  [INVALID_FILTER] Contains an assertion that is not valid or an unrecognized logical operator.

  [INVALID_SCOPE] The scope value is not valid.

  [NO_SUCH_OBJECT_CLASS] The managed-object class is not recognized.

  [NO_SUCH_OBJECT_INSTANCE] The managed-object instance is not recognized.

  [PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

  [SYNCHRONIZATION_NOT_SUPPORTED] This type of synchronization is not supported.

- One of the following *problem* values for a Library-Error:

  [BAD_ARGUMENT] An argument that was not valid was specified.

  [BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

  [BAD_CONTEXT] A context argument that was not valid was specified.

  [BAD_SESSION] A session that was not valid was specified.

  [MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

  [NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

  [SESSION_TERMINATED] The session is terminated and the results of an outstanding operation is no longer available.

[TIME_LIMIT_EXCEEDED]
The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). this OM attribute can be absent.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_bind(3)” on page 42.
- See “mp_delete_rsp(3)” on page 58.
- See “mp_receive(3)” on page 76.

mp_delete_rsp(3)

Purpose

Replies to a requested delete operation

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_delete_rsp( OM_private_object session,
                         OM_private_object context,
                         OM_object response,
                         OM_sint invoke_id );
```

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT.]

response Specifies the information supplied as the response to a delete operation. This information can be one of the following:

- An instance of a subclass of the OM class Linked-Reply-Argument, which expresses a partial result (linked reply). Each instance of the OM class CMIS-Linked-Reply-Argument contains only one of the following OM attributes:
  - *delete-Error*
  - *delete-Result*
  - *processing-Failure*
- An instance of a subclass of the OM class Delete-Result, which is supplied as the single reply to a delete operation. This instance indicates the successful completion of the operation. For a CMIS operation, this response is an instance of the OM class CMIS-Delete-Result.
- An instance of a subclass of the OM class Service-Error. For an instance of the OM class CMIS-Service-Error, one of the following *problem* values for a CMIS-Service-Error, as well as its associated *parameter*, can be sent as a reply:

  [ACCESS_DENIED] The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT] The managed-object instance is not a member of the specified class.

  [COMPLEXITY_LIMITATION] The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

  [INVALID_FILTER] Contains an assertion that is not valid or an unrecognized logical operator.

  [INVALID_SCOPE] The scope value is not valid.

[NO_SUCH_OBJECT_CLASS]
The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

[SYNCHRONIZATION_NOT_SUPPORTED]
This type of synchronization is not supported.

- The constant [MP_ABSENT_OBJECT], which indicates two possibilities:
  - The result is NULL (absent) because no object was selected.
  - This is the last response following a chain of linked replies.

invoke_id Specifies the invoke identification of the requested operation to which the reply applies.

Return Values

The mp_delete_rsp command returns a value to indicate whether the response was completed.

If successful, mp_delete_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_delete_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_ERROR] A service error that is not valid was specified.

[BAD_LINKED_REPLY]
A linked reply that is not valid was specified.

[BAD_RESULT] A result that is not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION]
The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use in this session.

[SESSION_TERMINATED]
The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_delete_req(3)” on page 55.
- See “mp_receive(3)” on page 76.

mp_error_message(3)

Purpose

Returns an error message describing a particular error

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_uint mp_error_message( MP_status error,
                          OM_uint length,
                          unsigned char *error_text_return );
```

Description

The requester provides a buffer-address parameter and a buffer-length parameter. The text of the error message is stored in the buffer of the requester and the length is returned.

Parameters

error Specifies the value that is returned from a function call.

length Specifies the length of the buffer. The error text buffer is an unsigned character array.

error_text_return Specifies a message describing the error. The error message text is terminated by a NULL character.

The text of the error is truncated if the length of the error-text buffer is less than the length of the text of the error message.

Return Values

The return value for mp_error_message indicates the length of the returned message.

Note: If *length* has the value 0 (zero) and if *error_text_return* is undefined, for example, if it has the NULL value, this function does not return any text. Instead, the function returns only the length required to contain the error message.

The mp_error_message function returns no errors. A default error message reports faulty parameters and other problems.

mp_event_report_req(3)

Purpose

Reports a notification emitted by a managed object

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_event_report_req( OM_private_object session,
                               OM_private_object context,
                               OM_object argument,
                               OM_private_object *result_return,
                               OM_sint *invoke_id_return );
```

Description

The mp_event_report_req function can be requested in a confirmed or nonconfirmed mode. A reply is expected in confirmed mode, but none is expected in nonconfirmed mode. An SNMP trap can be sent only in a nonconfirmed mode.

This function can be called in both synchronous and asynchronous modes.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the information supplied as the parameter of the notification to be reported. This information can be one of the following:

- For a CMIS event report, an instance of the OM class CMIS-Event-Report-Argument
- For an SNMP trap, an instance of the OM class SNMP-Trap-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, when the service was requested in a confirmed mode, the result is one of the following:

- An instance of the OM class CMIS-Event-Report-Result
- The constant [MP_ABSENT_OBJECT], which indicates the absence of a result.

invoke_id_return Specifies the invoke identification of the initiated management operation when invoked in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined. It is applicable only in a confirmed mode.

Return Values

The return value for this function indicates whether the report operation was completed, if used in synchronous mode, or whether the report operation was initiated if used in asynchronous mode.

If successful, mp_event_report_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_event_report_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a CMIS-Service-Error:

  [INVALID_ARGUMENT_VALUE]
  The event argument value was out of range or otherwise inappropriate.

  [NO_SUCH_ARGUMENT] The event or action is not recognized.

  [NO_SUCH_EVENT_TYPE]
  The event is not recognized.

  [NO_SUCH_OBJECT_CLASS]
  The managed-object class is not recognized.

  [NO_SUCH_OBJECT_INSTANCE]
  The managed-object instance is not recognized.

  [PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

- One of the following *problem* values for a Library-Error:

  [BAD_ARGUMENT] An argument that was not valid was specified.

  [BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

  [BAD_CONTEXT] A context argument that was not valid was specified.

  [BAD_SESSION] A session that was not valid was specified.

  [MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

  [NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

  [SESSION_TERMINATED]
  The session is terminated and the results of an outstanding operation is no longer available.

  [TIME_LIMIT_EXCEEDED]
  The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). This OM attribute can be absent.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_bind(3)” on page 42.
- See “mp_event_report_rsp(3)” on page 65.
- See “mp_receive(3)” on page 76.

mp_event_report_rsp(3)

Purpose

Replies to a previously reported management notification

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_event_report_rsp( OM_private_object session,
                               OM_private_object context,
                               OM_object response,
                               OM_sint invoke_id );
```

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

response Specifies the information supplied as the response to a reported event. This information is one of the following:

- An instance of a subclass of the OM class Event-Report-Result, which indicates the successful completion of the operation. For a CMIS operation, the response is an instance of the OM class CMIS-Event-Report-Result.
- The constant [MP_ABSENT_OBJECT], which indicates the absence of a result and means that the notification was completed successfully.
- An instance of a subclass of the OM class Service-Error, which indicates the failure of the notification:
  - For a CMIS operation, an instance of the OM class CMIS-Service-Error. The following *problem* values for a CMIS-Service-Error, as well as its associated *parameter*, can be sent as a reply:

    [INVALID_ARGUMENT_VALUE]
    The event argument value was out of range or otherwise inappropriate.

    [NO_SUCH_ARGUMENT] The event or action is not recognized.

    [NO_SUCH_EVENT_TYPE]
    The event is not recognized.

    [NO_SUCH_OBJECT_CLASS]
    The managed-object class is not recognized.

    [NO_SUCH_OBJECT_INSTANCE]
    The managed-object instance is not recognized.

    [PROCESSING_FAILURE]
    A general failure was encountered during the processing of an operation.

invoke_id Specifies the returned invoke identification of the reported notification to which the response applies.

Return Values

The return value for this function indicates whether the response was completed.

If successful, mp_event_report_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_event_report_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS]	The OM class of an argument, result, linked-reply, or error is not supported for this operation.
[BAD_CONTEXT]	A context argument that is not valid was specified.
[BAD_ERROR]	A service error that is not valid was specified.
[BAD_LINKED-REPLY]	A linked reply that is not valid was specified.
[BAD_SESSION]	A session that was not valid was specified.
[MISCELLANEOUS]	A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.
[NO_SUCH_OPERATION]	The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.
[SESSION_TERMINATED]	The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_event_report_req(3)” on page 62.
- See “mp_receive(3)” on page 76.

mp_get_next_req(3)

Purpose

Retrieves the next SNMP management information

Note: Using the get-next function can prevent a manager session from being portable on both the ISO CMIS and SNMP environments. For this reason, it is recommended that you avoid using this facility.

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_get_next_req( OM_private_object session,
                           OM_private_object context,
                           OM_object argument,
                           OM_private_object *result_return,
                           OM_sint *invoke_id_return );
```

Description

This function supports the SNMP get-next operation. It can be used only when the SNMP package has been selected using mp_version.

This function is defined as a confirmed service and can be called in both synchronous and asynchronous modes. A reply is expected.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies the information supplied as the parameter of a get-next operation is an instance of the OM class SNMP-Get-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, the result is an instance of the OM class SNMP-Get-Result, which contains a list of the variables and the values that were read.

invoke_id_return Specifies the returned invoke identification of the management operation, when used in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined.

Return Values

The return value for this function indicates whether the operation was completed, if used in synchronous mode; or whether the operation was initiated, if used in asynchronous mode.

If successful, mp_get_next_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_get_next_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- Any *problem* value for a SNMP-Service-Error
- One of the following *problem* values for a Library-Error:

  [BAD_ARGUMENT] An argument that was not valid was specified.

  [BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

  [BAD_CONTEXT] A context argument that was not valid was specified.

  [BAD_SESSION] A session that was not valid was specified.

  [MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

  [NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

  [SESSION_TERMINATED] The session is terminated and the results of an outstanding operation is no longer available.

  [TIME_LIMIT_EXCEEDED] The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). This OM attribute can be absent.

  [TOO_MANY_OPERATIONS] Additional management operations cannot be performed, until at least one asynchronous operation has been completed.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_get_rsp(3)” on page 72.
- See “mp_bind(3)” on page 42.
- See “mp_receive(3)” on page 76.

mp_get_req(3)

Purpose

Retrieves management information

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_get_req( OM_private_object session,
                      OM_private_object context,
                      OM_object argument,
                      OM_private_object *result_return,
                      OM_sint *invoke_id_return );
```

Description

- This service is defined as a confirmed service. A reply is expected.
- This function can be called in both synchronous and asynchronous modes. When used in asynchronous mode, the results of this operation can be discarded locally (through mp_abandon, as with other asynchronous calls). The remote operation also can be terminated (through mp_cancel_get_req).

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies an OM object that provides the information about which attributes are to be retrieved. This parameter is an instance of a subclass of the OM class Get-Argument:

- For a CMIS get operation, the supplied parameter is an instance of the OM class CMIS-Get-Argument.
- For an SNMP get operation, the supplied parameter is an instance of the OM class SNMP-Get-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, the result can be one of the following:

- For a CMIS get operation:
  - An instance of the OM class CMIS-Get-Result.

  - An instance of the OM class Multiple-Reply, which is a set of instances of the OM class CMIS-Linked-Reply-Argument. Each instance of the OM class CMIS-Linked-Reply-Argument contains one of the following attributes:
    - *get-List-Error*
    - *get-Result*
    - *processing-Failure*
  - The constant [MP_ABSENT_OBJECT], which indicates the absence of a result, if no managed objects were selected for the operation.
- For an SNMP get operation, an instance of the OM class SNMP-Get-Result, which contains a list of the variables and their values.

invoke_id_return Specifies the returned invoke identification of the management operation, when used in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined.

Return Values

The return value for this function indicates whether the action was completed, if used in synchronous mode; or whether it was initiated, if used in asynchronous mode.

If successful, mp_get_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_get_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- Any *problem* value for an SNMP-Service-Error
- One of the following *problem* values for a CMIS-Service-Error:

  [ACCESS_DENIED]
  The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT]
  The managed-object instance is not a member of the specified class.

  [COMPLEXITY_LIMITATION]
  The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

  [GET_LIST_ERROR]
  One or more attribute values were not read.

  [INVALID_FILTER]
  Contains an assertion that is not valid or an unrecognized logical operator.

  [INVALID_SCOPE]
  The scope value is not valid.

  [NO_SUCH_OBJECT_CLASS]
  The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[OPERATION_CANCELLED]
The get operation was canceled by a cancel-get operation, and no further attribute values will be returned by this invocation of the get service.

[PROCESSING_FAILURE]
When processing the operation, a general failure was encountered after the partial results were sent.

[SYNCHRONIZATION_NOT_SUPPORTED]
This type of synchronization is not supported.

- One of the following *problem* values for a Library-Error:

[BAD_ARGUMENT] An argument that was not valid was specified.

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

[SESSION_TERMINATED]
The session is terminated and the results of an outstanding operation is no longer available.

[SIZE_LIMIT_EXCEEDED]
The maximum number of linked responses, about which the requested service should return information, has been reached.

[TIME_LIMIT_EXCEEDED]
The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). This OM attribute can be absent.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_bind(3)” on page 42.
- See “mp_cancel_get_req(3)” on page 44.
- See “mp_get_rsp(3)” on page 72.
- See “mp_receive(3)” on page 76.

mp_get_rsp(3)

Purpose

Replies to a requested get operation or get-next operation

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_get_rsp( OM_private_object session,
                      OM_private_object context,
                      OM_object response,
                      OM_sint invoke_id );
```

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT].

response Specifies the information supplied as the response to a get or get-next operation.

The response to a CMIS get operation can be one of the following:

- An instance of the OM class CMIS-Linked-Reply-Argument, which expresses a partial result (linked reply). This parameter contains only one of the following OM attributes:
  - *get-List-Error*
  - *get-Result*
  - *processing-Failure*
- An instance of the OM class CMIS-Get-Result, which is the information supplied as the single reply to a CMIS get operation. It indicates the successful completion of the operation.
- An instance of the OM class CMIS-Service-Error, which indicates either that the operation has failed or that it has been canceled.

  One of the following *problem* values for a CMIS-Service-Error, as well as its associated *parameter*, can be sent as a reply:

  [ACCESS_DENIED] The request operation was not performed due to security reasons.

  [CLASS_INSTANCE_CONFLICT] The managed-object instance is not a member of the specified class.

  [COMPLEXITY_LIMITATION] The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.

  [GET_LIST_ERROR] One or more attribute values were not read.

[INVALID_FILTER] Contains an assertion that is not valid or an unrecognized logical operator.

[INVALID_SCOPE] The scope value is not valid.

[NO_SUCH_OBJECT_CLASS]
The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[OPERATION_CANCELLED]
The get operation was cancelled by a cancel-get operation, and no further attribute values will be returned by this invocation of the get service.

[PROCESSING_FAILURE]
A general failure was encountered during processing of the operation.

[SYNCHRONIZATION_NOT_SUPPORTED]
The type of synchronization is not supported.

- The constant [MP_ABSENT_OBJECT], which indicates two possibilities:
  - The result is NULL (absent) because no object was selected.
  - This is the last response following a chain of linked replies.

The response to an SNMP get operation or an SNMP get-next operation can be one of the following:

- An instance of the OM class SNMP-Get-Result, which indicates the successful completion of the SNMP get operation or get-next operation. The OM attribute *var-Bind-List* contains the list of variables with the values that were read.
- An instance of the OM class SNMP-Service-Error, which indicates the failure of the operation. Any *problem* value for an SNMP-Service-Error, including its associated *parameter*, can be sent as a reply.

invoke_id Specifies the invoke identification of the requested operation to which the reply applies.

Return Values

The mp_get_rsp command returns a value to indicate whether the response was completed.

If successful, mp_get_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_get_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error.
- Any *problem* value for a System-Error.
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_ERROR] A service error that is not valid was specified.

[BAD_LINKED-REPLY] A linked reply that is not valid was specified.

[BAD_RESULT] A result that is not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION] The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use in this session.

[SESSION_TERMINATED] The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_get_req(3)” on page 69.
- See “mp_get_next_req(3)” on page 67.
- See “mp_receive(3)” on page 76.

mp_initialize(3)

Purpose

Initializes the XOM workspace

Syntax

```
#include <xom.h>
#include <xmp.h>

OM_workspace mp_initialize( void );
```

Description

- This function performs any necessary initialization of the API, including the creation of a workspace. This function also performs any initialization of the API with the underlying MIS provider.
- This function must be called before any other management interface functions are called.
- This function can be called several times. In this case, each call returns a workspace that is distinct from other workspaces created by mp_initialize but not yet deleted by mp_shutdown.

Return Values

If successful, mp_initialize returns a handle to a workspace in which OM objects can be created and manipulated. Only objects that have been created in this workspace can be used as parameters to the other management interface functions. If unsuccessful, mp_initialize returns NULL.

Related Information

- See “mp_shutdown(3)” on page 87.

mp_receive(3)

Purpose

Retrieves the parameter of a management operation or notification; retrieves the partial result (linked reply) or the complete result of an asynchronous management operation or notification

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_receive( OM_private_object session,
                      OM_sint *primitive_return,
                      OM_sint *mode_return,
                      OM_sint *completion_flag_return,
                      MP_status *operation_notification_status_return,
                      OM_private_object *result_or_argument_return,
                      OM_sint *invoke_id_return );
```

Parameters

session Specifies the management session against which this management operation or notification is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

This function does not report any *problem* values for a Communications-Error or a Service-Error as return values. Any such errors, which relate to the completed asynchronous operation or notification, are reported in *operation_notification_status_return*.

primitive_return

Specifies one of the following service primitives:

Table 12. Service Primitives

Indicate	Confirm
MP_GET_IND	MP_GET_CNF
MP_GET_NEXT_IND	MP_GET_CNF
MP_SET_IND	MP_SET_CNF
MP_ACTION_IND	MP_ACTION_CNF
MP_CREATE_IND	MP_CREATE_CNF
MP_DELETE_IND	MP_DELETE_CNF
MP_EVENT_REPORT_IND	MP_EVENT_REPORT_CNF
MP_CANCEL_GET_IND	MP_CANCEL_GET_CNF

It determines the management operation or notification of this result or parameter.

This result is valid only if *completion_flag_return* has the value [MP_T_COMPLETED], [MP_T_INCOMING], or [MP_T_PARTIAL].

mode_return

When the value is [MP_T_CONFIRMED], the invoked management operation or the reported management notification has to be confirmed. A reply is expected. When the value is [MP_T_NON_CONFIRMED], the requested management service is not to be confirmed.

This result is valid only if *completion_flag_return* has the value [MP_T_INCOMING].

completion_flag_return

Takes one of the following values to indicate the status of outstanding asynchronous operations or notifications:

- [MP_T_COMPLETED]

  At least one outstanding asynchronous operation or notification has been completed, and its result (positive as well as negative) is available.

- [MP_T_INCOMING]

  An incoming operation or notification indication is available.

- [MP_T_NOTHING]

  There are neither outstanding asynchronous operations or notifications, nor incoming indications.

- [MP_T_OUTSTANDING]

  There are outstanding asynchronous operations or notifications, but none has been completed (even partially) yet.

- [MP_T_PARTIAL]

  At least one outstanding asynchronous operation has been processed, and a partial result is available.

This result is valid only if the return value of the function is [MP_SUCCESS]. The validity of the other results in that case is given in the following table:

Table 13. Validity of Completion Flag Values

Completion Flag Value	Primitive	Mode	Operation or Notification Status	Result or Parameter	Invoke ID
[MP_T_COMPLETED]	Yes (*)	No	Yes	Yes (*)	Yes
[MP_T_INCOMING]	Yes	Yes	No	Yes	Yes
[MP_T_NOTHING]	No	No	No	No	No
[MP_T_OUTSTANDING]	No	No	No	No	No
[MP_T_PARTIAL]	Yes (*)	No	Yes	Yes (*)	Yes

Note: An asterisk (*) indicates that the result is valid only if *operation_notification_status_return* has the value [MP_SUCCESS].

operation_notification_status_return

This result is valid only if *completion_flag_return* has the value [MP_T_COMPLETED] or [MP_T_PARTIAL].

The *operation_notification_status_return* parameter takes one of the following values to indicate whether the asynchronous management operation or notification was executed successfully:

- The constant [MP_SUCCESS], if the operation or notification was successful

- An instance of the OM class CMIS-Linked-Reply-Argument, containing the partial result of an asynchronous operation.
- One of the following error values, if an error occurred during execution of the operation or notification:
  - The constant [MP_NO_WORKSPACE]
  - The constant [MP_INVALID_SESSION]
  - The constant [MP_INSUFFICIENT_RESOURCES]
  - Any *problem* value for a Communications-Error.
  - Any *problem* value for a SNMP-Service-Error
  - One of the *problem* values for a CMIS-Service-Error. The possible error values are listed for each operation in the following table.

Table 14. Valid CMIS-Service-Error Values for each Confirm Primitive

Error Values	ACT	CAN	CRE	DEL	EVE	GET	SET
[ACCESS_DENIED]	yes		yes	yes		yes	yes
[CLASS_INSTANCE_CONFLICT]	yes		yes	yes		yes	yes
[COMPLEXITY_LIMITATION]	yes			yes		yes	yes
[DUPLICATE_MANAGED_OBJECT]			yes				
[GET_LIST_ERROR]						yes	
[INVALID_ARGUMENT_VALUE]	yes				yes		
[INVALID_ATTRIBUTE_VALUE]			yes				
[INVALID_FILTER]	yes			yes		yes	yes
[INVALID_OBJECT_INSTANCE]			yes				
[INVALID_SCOPE]	yes			yes		yes	yes
[MISSING_ATTRIBUTE_VALUE]			yes				
[MISTYPED_OPERATION]		yes					
[NO_SUCH_ACTION]	yes						
[NO_SUCH_ARGUMENT]	yes				yes		
[NO_SUCH_ATTRIBUTE]			yes				
[NO_SUCH_EVENT_TYPE]					yes		
[NO_SUCH_INVOKE_ID]		yes					
[NO_SUCH_OBJECT_CLASS]	yes		yes	yes	yes	yes	yes
[NO_SUCH_OBJECT_INSTANCE]	yes		yes	yes	yes	yes	yes
[NO_SUCH_REFERENCE_OBJECT]			yes				
[OPERATION_CANCELLED]						yes	
[PROCESSING_FAILURE]	yes	yes	yes	yes	yes	yes	yes
[SET_LIST_ERROR]							yes
[SYNCHRONIZATION_NOT_SUPPORTED]	yes			yes		yes	yes

Note: The confirm primitives are abbreviated as follows in the table headers:

ACT = MP_ACTION_CNF
CAN = MP_CANCEL_GET_CNF
CRE = MP_CREATE_CNF
DEL = MP_DELETE_CNF
EVE = MP_EVENT_REP_CNF
GET = MP_GET_CNF
SET = MP_SET_CNF

result_or_argument_return

Specifies the result of the (partially) completed asynchronous operation or notification, the parameter of the invoked management operation or the parameter of the reported management notification.
The value of this result or parameter is the constant [MP_ABSENT_OBJECT] in each of the following cases:

- The operation was one which does not return a result (for example, mp_cancel_get).
- No object was selected for the operation.
- This is the last response following a chain of linked replies.

Otherwise, the OM class of the OM object is the OM class of the result (or partial result) of the asynchronous operation or notification, or the OM class of the parameter of the invoked operation or reported notification. The particular class of the OM object can be determined by using the OM functions.

This result is valid if the return value of the function is [MP_SUCCESS]; *completion_flag_return* has the value [MP_T_COMPLETED] or [MP_T_PARTIAL]; and *operation_notification_status_return* has the value [MP_SUCCESS].

This result is also valid if the return value of the function is [MP_SUCCESS], and *completion_flag_return* has the value [MP_T_INCOMING].

invoke_id_return

Specifies the invoke identification or notification or the operation for which an error, a result, or a parameter is being returned.

This result is valid only if the return value of the function is [MP_SUCCESS], and *completion_flag_return* has the value [MP_T_COMPLETED], [MP_T_PARTIAL], or [MP_T_INCOMING].

Return Values

If successful, mp_receive returns the constant [MP_SUCCESS]. If unsuccessful, mp_receive returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[SESSION_TERMINATED]
The session is terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.

mp_set_req(3)

Purpose

Modifies the attribute values of managed objects

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_set_req( OM_private_object session,
                      OM_private_object context,
                      OM_object argument,
                      OM_private_object *result_return,
                      OM_sint *invoke_id_return );
```

Description

- The CMIS service can be requested in a confirmed mode or a nonconfirmed mode. A reply is expected in a confirmed mode, while none is expected in a nonconfirmed mode. The SNMP service can be requested only in a confirmed mode.
- This function can be called in both synchronous and asynchronous modes.

Parameters

session Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context Specifies the management context to be used for this operation. This argument is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT]. This parameter defines the modes of operation and the possible addressing and access-control parameters.

argument Specifies an OM object that provides the information about which attributes are to be updated. This parameter is an instance of a subclass of the OM class Set-Argument:

- For a CMIS set operation, the supplied argument is an instance of the OM class CMIS-Set-Argument.
- For an SNMP set operation, the supplied argument is an instance of the OM class SNMP-Set-Argument.

result_return Specifies a private object. Upon successful completion of a synchronous call, when the operation was requested in a confirmed mode, the result can be one of the following:

- For a CMIS set operation:
  - An instance of the OM class CMIS-Set-Result.
  - An instance of the OM class Multiple-Reply, which is a set of instances of a subclass of the OM class CMIS-Linked-Reply-Argument. This instance contains one of the following OM attributes:
    - *processing-Failure*
    - *set-List-Error*
    - *set-Result*
- For an SNMP set operation, the result is an instance of the OM class SNMP-Set-Result, which contains the requested list of variables with the values that were modified.

Otherwise, in nonconfirmed mode, no results are expected. The constant [MP_ABSENT_OBJECT] denotes the absence of a result.

invoke_id_return Specifies the invoke identification of the initiated management operation, when invoked in asynchronous mode. This value allows the results retrieved through the mp_receive function to be matched with the original request. In synchronous mode, this parameter is undefined. It is applicable only in a confirmed mode.

Return Values

The return value for this function indicates whether the action was completed, if used in synchronous mode; or whether it was initiated, if used in asynchronous mode.

If successful, mp_set_req returns the constant [MP_SUCCESS]. If unsuccessful, mp_set_req returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- Any *problem* value for an SNMP-Service-Error
- One of the following *problem* values for a CMIS-Service-Error:

[ACCESS_DENIED]	The request operation was not performed due to security reasons.
[CLASS_INSTANCE_CONFLICT]	The managed-object instance is not a member of the specified class.
[COMPLEXITY_LIMITATION]	The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.
[INVALID_FILTER]	Contains an assertion that is not valid or an unrecognized logical operator.

[INVALID_SCOPE] The scope value is not valid.

[NO_SUCH_OBJECT_CLASS] The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

[SET_LIST_ERROR] One or more attribute values were not modified.

[SYNCHRONIZATION_NOT_SUPPORTED]
This type of synchronization is not supported.

- One of the following *problem* values for a Library-Error:

[BAD_ARGUMENT] An argument that was not valid was specified.

[BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

[BAD_CONTEXT] A context argument that was not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use on the session.

[SESSION_TERMINATED]
The session is terminated and the results of an outstanding operation are no longer available.

[SIZE_LIMIT_EXCEEDED]
The maximum number of linked responses, about which the requested service should return information, has been reached.

[TIME_LIMIT_EXCEEDED]
The maximum elapsed time, within which the requested service must be provided, has been reached. The OM attribute parameter specifies a Multiple-Reply object, which contains received partial results. The syntax is object(Multiple-Reply). This OM attribute can be absent.

[TOO_MANY_OPERATIONS.]
Additional management operations can not be performed until at least one asynchronous operation has been completed.

Related Information

- See “mp_abandon(3)” on page 34.
- See “mp_bind(3)” on page 42.
- See “mp_receive(3)” on page 76.
- See “mp_set_rsp(3)” on page 84.

mp_set_rsp(3)

Purpose

Replies to a requested set operation

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_set_rsp( OM_private_object session,
                      OM_private_object context,
                      OM_object response,
                      OM_sint invoke_id );
```

Parameters

session

Specifies the management session against which this operation is performed. This is the private object of the OM class Session that was previously returned from mp_bind.

context

Specifies the management context to be used for this operation. This parameter is a private object of the OM class Context, or the constant [MP_DEFAULT_CONTEXT].

response

Specifies the information supplied as the response to the requested set operation.

The response to a CMIS set operation can be one of the following:

- An instance of the OM class CMIS-Linked-Reply-Argument, which indicates a partial result (linked reply). This instance contains one of the following OM attributes:
  - *processing-Failure*
  - *set-List-Error*
  - *set-Result*
- An instance of the OM class CMIS-Set-Result, which is supplied as the single reply to a CMIS set operation and indicates the successful completion of the operation.
- An instance of the OM class CMIS-Service-Error, which indicates the failure of the operation. One of the following *problem* values for a CMIS-Service-Error, including its associated *parameter*, can be sent as a reply:

[ACCESS_DENIED]	The request operation was not performed due to security reasons.
[CLASS_INSTANCE_CONFLICT]	The managed-object instance is not a member of the specified class.
[COMPLEXITY_LIMITATION]	The requested operation was not performed, because an OM attribute, such as scope, filter, or synchronization was too complex.
[INVALID_FILTER]	Contains an assertion that is not valid or an unrecognized logical operator.
[INVALID_SCOPE]	The scope value is not valid.

[NO_SUCH_OBJECT_CLASS] The managed-object class is not recognized.

[NO_SUCH_OBJECT_INSTANCE]
The managed-object instance is not recognized.

[PROCESSING_FAILURE] A general failure was encountered during the processing of an operation.

[SET_LIST_ERROR] One or more attribute values were not modified.

[SYNCHRONIZATION_NOT_SUPPORTED]
This type of synchronization is not supported.

- The constant [MP_ABSENT_OBJECT], which indicates two possibilities:
  - The result is NULL (absent) because no object was selected.
  - This is the last response following a chain of linked replies.

  The response to an SNMP set operation can be one of the following:

  - An instance of the OM class SNMP-Set-Result, which indicates the successful completion of the SNMP set operation.

    The single OM attribute *var-Bind-List* contains the requested list of variables with the corresponding values that were modified.

  - An instance of the OM class SNMP-Service-Error, which indicates the failure of the operation. Any *problem* value for an SNMP-Service-Error, along with its associated *parameter*, can be sent as a reply.

invoke_id
Specifies the invoke identification of the requested operation to which the reply applies.

Return Values

The mp_set_rsp command returns a value to indicate whether the response was completed. If successful, mp_set_rsp returns the constant [MP_SUCCESS]. If unsuccessful, mp_set_rsp returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a Communications-Error
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

  [BAD_CLASS] The OM class of an argument, result, linked-reply, or error is not supported for this operation.

  [BAD_CONTEXT] A context argument that was not valid was specified.

  [BAD_ERROR] A service error that is not valid was specified.

  [BAD_LINKED_REPLY]
  A linked reply that is not valid was specified.

  [BAD_RESULT] A result that is not valid was specified.

  [BAD_SESSION] A session that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[NO_SUCH_OPERATION]
The library has no knowledge of the designated operation and notification in progress, or the response does not match the invoked operation and notification.

[NOT_SUPPORTED] An attempt was made to use an option that is not available or was not agreed upon for use in this session.

[SESSION_TERMINATED]
The session was terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_receive(3)” on page 76.
- See “mp_set_req(3)” on page 81.

mp_shutdown(3)

Purpose

Frees or discards a workspace

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status  mp_shutdown( OM_workspace workspace );
```

Description

- This function deletes a workspace established by mp_initialize and enables the service to release resources.
- No other management interface functions should be called after this function, except for mp_initialize.
- All the remaining open sessions are closed, all the remaining private OM objects are deleted, and the workspace is deleted. Service-generated public objects must be deleted by calling om_delete explicitly, since they are not affected by mp_shutdown.

Parameters

workspace
: Specifies the handle to the workspace.

Return Values

If successful, mp_shutdown returns the constant [MP_SUCCESS]. If unsuccessful, mp_shutdown returns one of the following error codes.

Error Codes

- [MP_NO_WORKSPACE]
- [MP_INSUFFICIENT_RESOURCES]

Related Information

- See “mp_initialize(3)” on page 75.

mp_unbind(3)

Purpose

Terminates a management session

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_unbind( OM_private_object session );
```

Description

- By terminating a given management session, this function makes the parameter *session* unavailable for use with other interface functions (except mp_bind()). This means the results of any outstanding asynchronous operations, which were initiated using the given *session*, can no longer be received. (Such operations can be terminated prematurely.) For this reason, it is recommended that you process all outstanding asynchronous operations with mp_receive before using mp_unbind.
- The unbound session can be used again as an argument to mp_bind, possibly after modification by the XOM functions. When it is no longer required, the session must be deleted using the om_delete function call.

Parameters

session Specifies the management session, which is to be unbound. This is the private object of the OM class Session that was previously returned from mp_bind. The value of the *file-Descriptor* OM attribute is [MP_NO_VALID_FILE_DESCRIPTOR] if the function succeeds. The other OM attributes are unchanged.

Return Values

The mp_unbind command returns a value to indicate whether *session* was unbound successfully. If successful, mp_unbind returns the constant [MP_SUCCESS]. If unsuccessful, mp_unbind returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- One of the following *problem* values for a Library-Error:

[BAD_CLASS]	The OM class of an argument, result, linked-reply, or error is not supported for this operation.
[BAD_SESSION]	A session that was not valid was specified.
[MISCELLANEOUS]	A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[SESSION_TERMINATED]
: The session is terminated and the results of an outstanding operation are no longer available.

This function does not return a Communications-Error or any management service errors.

Related Information

- See “mp_bind(3)” on page 42.

mp_version(3)

Purpose

Negotiates features of the interface and service

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_version( OM_workspace workspace,
                      MP_feature feature_list[] );
```

Description

This function negotiates features of the XMP API, which are represented by object identifiers. The CMIS package and the SNMP package, specified in Chapter 4, “XMP API Management Service Packages” on page 963, and the Management Contents packages, specified in Chapter 5, “XMP API Management Contents Packages” on page 1033, are negotiable features.

Parameters

workspace Specifies the handle to the workspace.

feature_list Specifies an ordered sequence of features, each represented by an object identifier. The sequence is terminated by an object identifier having no components (a length of 0 (zero) and any value of the data pointer in the C language representation).

Return Values

activated If the function completed successfully, the result contains an ordered sequence of Boolean values, with the same number of elements as the *feature_list*. If *true*, each value indicates that the corresponding feature is now part of the interface. If *false*, each value indicates that the corresponding feature is not available.

In the C language binding, the result is combined with the *feature_list* argument as a single array of structures of type MP_feature, which is defined as:

```
typedef struct {
                  OM_object_identifier feature;
                  OM_boolean activated;
                } MP_feature;
```

The mp_version command returns a value to indicate whether the response was completed. If successful, mp_version returns the constant [MP_SUCCESS]. If unsuccessful, mp_version returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a System-Error
- The *miscellaneous* Library-Error

The mp_version command does not return a Communications-Error or any management service error.

Implementation Specifics

Although it should never fail in a production system, you may experience problems with mp_version() calls in a development environment, where libraries, library paths, object identifiers, and other elements are constantly changing. If you experience problems with mp_version– that is, if it is unable to load a package– read the following information.

When a program calls mp_version to request that an OM package be loaded, XMP will do the following:

Step 1. Read the configuration file /usr/OV/conf/xmpcfg.dat. Locate the oid in the configuration and get the name of the corresponding library file.

Step 2. Call the AIX load() subroutine to bring this library into memory.

The following errors can occur during this process:

- The configuration file does not exist.

  Execute the command **ls -l /usr/OV/conf/xmpcfg.dat** and check whether the file is there. If it is not you will have to reinstall the product or get one from backup.

- The configuration file cannot be read.

  This may be a file permission problem. Log in as the userid that executes the program that calls mp_version and execute the command **cat /usr/OV/conf/xmpcfg.dat** to see whether the file can be read. If the file is there but cannot be read, log in as root and execute the command **chmod 444 /usr/OV/conf/xmpcfg.dat**.

- The oid in the mp_version package does not match the one in the configuration file.

  This is most likely caused by including a header file that is not compatible with the configuration file in use. Check whether the oid in the call to mp_version matches one of those in the configuration file. If not, recompile your program with the correct header file or get a new library.

- The library cannot be loaded

  If your library is present, is readable, and matches an entry in the configuration file, but mp_version cannot load a package from it, there may be a problem in your library path. If the environment variable LIBPATH is defined in the context of the process, XMP lets the operating system resolve the path to the library file to be loaded. The system will then check all the directories in the LIBPATH to locate the library. See the description of the load() routine for more details.

  If LIBPATH is not defined, XMP explicitly adds /usr/OV/lib in the front of the library name, so that the library will be loaded from there.

  If you set your LIBPATH make sure that the directory where the libraries reside is part of the LIBPATH or XMP will not be able to load your OM packages.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_initialize(3)” on page 75.

mp_wait(3)

Purpose

Suspends the caller until a management message is available from one or more bound sessions

Syntax

```
#include <xom.h>
#include <xmp.h>

MP_status mp_wait( MP_waiting_sessions bound_session_list[]
                   OM_workspace workspace,
                   OM_uint32 timeout );
```

Description

Once mp_wait indicates that there are messages available, you should call mp_receive repeatedly until it returns MP_T_NOTHING or MP_T_OUTSTANDING, that is, until you have received all messages available) before calling mp_wait again.

Parameters

bound_session_list Specifies a list of management sessions upon which to wait. Each list value is a private object of the OM class Session, and a flag that indicates if there are any messages in that session. The last value must be the constant [MP_DEFAULT_SESSION].

workspace Specifies the workspace (obtained from a call from mp_initialize), in which an MP_status OM object is created if the return value is something other than the constant [MP_SUCCESS]. Sessions specified in the *bound_session_list* do not need to be from this workspace.

timeout Specifies the maximum number of milliseconds for which the requester is suspended before obtaining a response, when there are no messages from the list of sessions. A value of zero specifies an indefinite timeout.

Return Values

The mp_wait command returns a value to indicate whether the action was completed. If successful, mp_wait returns the following return values:

- The constant [MP_SUCCESS]. A successful completion means either that a message is available from a session or that the timeout limit has been reached. The mp_receive function must be called to determine whether a message is available.
- Activated(OM_boolean). If the function was completed successfully, this result contains an ordered sequence of Boolean values, with the same number of elements as the **bound_session_list**. If true, each value indicates that the corresponding Session has data waiting in queue. If false, each value indicates that the corresponding Session does NOT have data waiting in queue.

In the C binding, this result is combined with the bound_session_list argument as a single array of structures of type MP_waiting_sessions, which is defined as:

```
typedef struct {
                  OM_private_object bound_session;
                  OM_boolean activated;
                } MP_waiting_sessions;
```

If unsuccessful, mp_wait returns one of the following error codes.

Error Codes

- The constant [MP_NO_WORKSPACE]
- The constant [MP_INVALID_SESSION]
- The constant [MP_INSUFFICIENT_RESOURCES]
- Any *problem* value for a System-Error
- One of the following *problem* values for a Library-Error:

[BAD_ADDRESS] An address that is not valid was specified.

[BAD_SESSION] A session that was not valid was specified.

[BAD_WORKSPACE] A workspace argument that was not valid was specified.

[MISCELLANEOUS] A miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.

[SESSION_TERMINATED] The session is terminated and the results of an outstanding operation are no longer available.

Related Information

- See “mp_bind(3)” on page 42.
- See “mp_initialize(3)” on page 75.
- See “mp_receive(3)” on page 76.

nvCollectionAdd(3)

Purpose

Defines new collections of objects

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int nvCollectionAdd(char *name,
                    char *desc,
                    char *rule,
                    int  force);
```

Description

The nvCollectionAdd function adds a new collection of objects to the Collection Facility. You provide a name for the collection, a description, and the rule for determining which objects fit into the collection. Note that this call can be CPU-intensive if the collection is large.

Parameters

name
Specifies the name of the collection

desc
Provides a description of what is in the collection

rule
Provides a rule for determining which objects fit into the collection. The rule is specified using one of the following constructs:

IN 'object1 object2...'
: Selects a list of objects, separated by blanks. If more than one object is specified, the list must be enclosed in single quotes.

field op value
: Selects objects in the object database that meet the criteria specified in this expression, such as `isRouter=true`.

  field is a field in the object database, such as `isRouter`.

  op is one of the following logical operators:

  - =
  - !=
  - <
  - >
  - <=
  - >=

  If *field* or *value* has spaces in the text (such as `'SNMP sysObjectId'`), enclose the text in single quotes.

IN_SUBNET Selects objects within the specified subnet

IN_COLLECTION Selects objects in another previously defined collection.

These rules can be made more complex by joining simple rules with AND, &&, OR, ||, and ! operators within parentheses. For example, you might specify the following rule:

```
((isRouter=true) AND ((IN_SUBNET 9.67.96.0) || (IN_SUBNET 9.60.100.0)))
```

force
Forces the add function if a dependent collection does not exist.

Return Values

If successful, nvCollectionAdd returns a value of 0. If unsuccessful, nvCollectionAdd returns a value of –1.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_COLLECTION_EXISTS]
The collection cannot be created because another collection with that name already exists.

[NV_COLLECTION_INVALID_RULE]
For some reason, the rule cannot be processed. Check the Boolean logic and any other collection names you specified.

[NV_COLLECTION_FIELD_NOT_VALID]
The *field* specified in *field op value* is not a valid field.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the new collection information.

[NV_COLLECTION_DEPENDENCY_NOT_FOUND]
The collection being added uses another collection in its definition, and that collection cannot be found.

[NV_COLLECTION_OVw_ERROR] An error occurred while trying to add the collection to the object database.

Files

When compiling a program that uses nvCollectionAdd, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionAdd, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionAddCallback(3)” on page 98.
- See “nvCollectionError(3)” on page 103.

nvCollectionAddCallback(3)

Purpose

Registers procedures to process collection facility events

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int nvCollectionAddCallback(nvCollectionEvent event,
                            nvCollectionCallback callback,
                            void *callback_data);
```

Description

The nvCollectionAddCallback function adds an application callback procedure for handling the events associated with the collection facility. Events include:

- Adding a new collection
- Deleting a collection
- Modifying the description or rule for a collection
- Adding or removing an object from an existing collection
- Collection errors

You should register for the error event if you are registering for any other events. This notifies the program that the connection to the collection facility daemon (nvcold) has been lost.

Parameters

event
Specifies the collection facility event that should be processed. Collection facility events are as follows:

NV_COLLECTION_ADDED
New collection was added

NV_COLLECTION_DELETED
Collection was deleted

NV_COLLECTION_DESC_MODIFIED
The description for a collection was modified

NV_COLLECTION_RULE_MODIFIED
The rule for defining which objects belong in the collection was modified

NV_COLLECTION_OBJ_ADDED
An object was added to a collection

NV_COLLECTION_OBJ_DELETED
An object was deleted from a collection

NV_COLLECTION_ERROR
An unspecified error occurred.

callback
Specifies a procedure to invoke for the specified event.

callback_data
Specifies a pointer to data that is passed to the callback procedure.

Return Values

If successful, the nvCollectionAddCallback return value is 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

[NV_COLLECTION_SUCCESS]	Successful completion.
[NV_OUT_OF_MEMORY]	There is not enough memory to complete the operation.

Files

When compiling a program that uses nvCollectionAddCallback, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionAddCallback, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionAdd(3)” on page 95.

nvCollectionDelete(3)

Purpose

Deletes a collection definition

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int nvCollectionDelete(char *name);
```

Description

The nvCollectionDelete function deletes a collection definition.

Parameters

name
Specifies the name of the collection

Return Values

If successful, nvCollectionDelete returns a value of 0. If unsuccessful, nvCollectionDelete returns a value of –1.

Error Codes

NV_COLLECTION_SUCCESS — Successful operation.

NV_COLLECTION_DOES_NOT_EXIST
The specified collection is not defined and cannot be deleted.

[NV_COLLECTION_DEPENDENCY_EXISTS]
Another collection is dependent on this collection. This collection is specified as part of another collection's definition.

Files

When compiling a program that uses nvCollectionDelete, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionDelete, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionAddCallback(3)” on page 98.
- See “nvCollectionError(3)” on page 103.

nvCollectionDone(3)

Purpose

Closes a connection to the collection facility server

Related Functions

nvCollectionXDone

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int    nvCollectionDone();

int    nvCollectionXDone();
```

Description

The nvCollectionDone function closes the connection to the collection facility server.

The nvCollectionXDone function should be used if you used nvCollectionXOpen to open the connection to the server.

Files

When compiling a program that uses nvCollectionDone or nvCollectionXDone, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionDone or nvCollectionXDone, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionOpen(3)” on page 119.
- See “nvCollectionXOpen(3)” in “nvCollectionOpen(3)” on page 119.

nvCollectionError(3)

Purpose

Returns the error code set by the last collection facility API call

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int    nvCollectionError();
```

Description

The nvCollectionError function returns the error value set by the previous collection facility API call. It can be tested immediately after a failed collection facility call (either a –1 or a NULL pointer) to determine the exact reason for the failure.

Examples

The following code example illustrates the way nvCollectionError can be used with nvCollectionErrorMsg:

```
if (nvCollectionOpen() < 0) {
    fprintf(stderr, "foo: %s\n", nvCollectionErrorMsg(nvCollectionError()));
    exit(1);
}
```

Files

When compiling a program that uses nvCollectionError, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionError, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionErrorMsg(3)” on page 104.

nvCollectionErrorMsg(3)

Purpose

Returns a textual description of a collection facility API error code

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

char  *  nvCollectionErrorMsg(int error);
```

Description

The nvCollectionErrorMsg function maps a collection facility API error code to a string that contains text describing the meaning of the specified error code.

Examples

The following code example illustrates the way nvCollectionError can be used with nvCollectionErrorMsg:

```
if (nvCollectionOpen() < 0) {
    fprintf(stderr, "foo: %s\n", nvCollectionErrorMsg(nvCollectionError()));
    exit(1);
}
```

Files

When compiling a program that uses nvCollectionErrorMsg, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionErrorMsg, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionError(3)” on page 103.

nvCollectionEvaluate(3)

Purpose

Evaluates a rule and returns a list of objects that fit the rule

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

OVwObjectIdList * nvCollectionEvaluate(char * rule);
```

Description

The nvCollectionEvaluate function can be used to determine which objects in a network will be included in collections that use the specified rule.

Parameters

rule

Provides a rule for determining which objects fit into the collection. The rule is specified using one of the following constructs:

IN 'object1 object2...'
: Selects a list of objects, separated by blanks. If more than one object is specified, the list must be enclosed in single quotes.

field op value
: Selects objects in the object database that meet the criteria specified in this expression, such as `isRouter=true`.

  field is a field in the object database, such as `isRouter`.

  op is one of the following logical operators:

  - =
  - !=
  - <
  - >
  - <=
  - >=

  If *field* or *value* has spaces in the text (such as `'SNMP sysObjectId'`), enclose the text in single quotes.

IN_SUBNET
: Selects objects within the specified subnet

IN_COLLECTION
: Selects objects in another previously defined collection.

These rules can be made more complex by joining simple rules with AND, &&, OR, ||, and ! operators within parentheses. For example, you might specify the following rule:

```
((isRouter=true) AND ((IN_SUBNET 9.67.96.0) || (IN_SUBNET 9.60.100.0)))
```

Return Values

If successful, nvCollectionEvaluate returns a pointer to an OVw object ID list. If unsuccessful, nvCollectionEvaluate returns NULL.

Error Codes

NV_COLLECTION_SUCCESS — Successful operation.

NV_COLLECTION_INVALID_RULE
: For some reason, the rule cannot be processed. Check the Boolean logic and any other collection names you specified.

NV_COLLECTION_FIELD_NOT_VALID
: One of the fields specified in the rule is invalid.

Files

When compiling a program that uses nvCollectionEvaluate, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionEvaluate, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

nvCollectionFreeDefn(3)

Purpose

Frees memory used for collection facility functions

Related Functions

nvCollectionFreeList
nvCollectionFreeChangeList

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

void  nvCollectionFreeDefn(nvCollectionDefn *defn);

void  nvCollectionFreeList(nvCollectionList *list);

void  nvCollectionFreeChangeList(nvCollectionChangeList *list);
```

Description

The nvCollectionFreeDefn function frees memory used for returning collection definitions.

The nvCollectionFreeList function frees memory used for returning lists of collections.

The nvCollectionFreeChangeList function frees memory used for returning lists of changes to collections.

Parameters

defn
Specifies a pointer to a collection definition.

list
Specifies a pointer to a collection list.

Files

When compiling a program that uses nvCollectionFreeDefn, nvCollectionFreeChangeList, or nvCollectionFreeList, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionFreeDefn, nvCollectionFreeChangeList, or nvCollectionFreeList, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

nvCollectionGetAllForObject(3)

Purpose

Obtains a list of all collections the specified object is a member of.

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

nvCollectionList *   nvCollectionGetAllForObject(OVwObjectId objectid);
```

Description

The nvCollectionGetAllForObject function returns a list of all collections the specified object is a member of.

Parameters

objectid
Specifies the object ID of the object for which you want information.

Return Values

If successful, nvCollectionGetAllForObject returns a list of collections. If unsuccessful, nvCollectionGetAllForObject returns NULL.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to complete the operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The specified collection is not defined.

Files

When compiling a program that uses nvCollectionGetAllForObject, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionGetAllForObject, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionError(3)” on page 103.

nvCollectionGetInfo(3)

Purpose

Obtains the description and rule defined for a collection

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int    nvCollectionGetInfo( char *name,
                            char **desc,
                            char **rule);
```

Description

The nvCollectionGetInfo function returns the description and the rule that have been defined for the specified collection. Because the return values for the description and the rule are dynamically allocated, you must free the strings when they are no longer needed.

Parameters

name
Specifies the collection for which you want information

desc
Specifies the address of a pointer to the description returned for the collection. This will point to the retrieved data.

rule
Specifies the address of a pointer to the rule returned for the collection. This will point to the retrieved data.

Return Values

If successful, nvCollectionGetInfo returns a value of 0. If unsuccessful, it returns a value of –1.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The specified collection is not defined.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the collection information.

Files

When compiling a program that uses nvCollectionGetInfo, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionGetInfo, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionError(3)” on page 103.
- See “nvCollectionFreeDefn(3)” on page 107.

nvCollectionGetTimestamp(3)

Purpose

Returns the last time a collection was updated.

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

time_t nvCollectionGetTimestamp(char * name);
```

Description

The nvCollectionGetTimestamp function can be used to determine the last time a collection was updated.

Parameters

name
Specifies the name of the collection for which a timestamp is to be returned.

Return Values

If successful, nvCollectionEvaluate returns a timestamp. If unsuccessful, nvCollectionGetAllForObject returns –1.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The collection specified does not exist.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the collection information.

Files

When compiling a program that uses nvCollectionGetTimestamp, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionGetTimestamp, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

nvCollectionIntersect(3)

Purpose

Finds the intersection of two collections

Related Functions

nvCollectionListIntersect

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

OVwObjectIdList * nvCollectionIntersect(char *name1,
                                        char *name2);

OVwObjectIdList * nvCollectionListIntersect(char *name1,
                                        OVwObjectIdList   *list);
```

Description

The nvCollectionIntersect function finds the intersection between two collections. A list is generated of all objects that are members in both collections.

The nvCollectionListIntersect function finds the intersection between a collection and an object ID list. A list is generated of all objects from the object list that are in the collection.

Parameters

name1
Specifies the name of the first collection to be intersected

name2
Specifies the name of the second collection to be intersected

list
Specifies a list of object IDs.

Return Values

If successful, nvCollectionIntersect and nvCollectionListIntersect return a list of object IDs. If unsuccessful, they return NULL.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The specified collection is not defined and cannot be intersected.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the collection information.

Files

When compiling a program that uses nvCollectionIntersect or nvCollectionListIntersect, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionIntersect or nvCollectionListIntersect, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionError(3)” on page 103.
- See “nvCollectionUnion(3)” on page 124.
- See “nvCollectionListUnion(3)” in “nvCollectionUnion(3)” on page 124.

nvCollectionListCollections(3)

Purpose

Obtains a list of all collections currently defined

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

nvCollectionList*   nvCollectionListCollections();
```

Description

The nvCollectionListCollections function returns a list of all collections that are currently defined. Because the return values for the description and the rule are dynamically allocated, you must use the nvCollectionFreeList function to free the strings when they are no longer needed.

Return Values

If successful, nvCollectionListCollections returns a list of defined collections. Otherwise, it returns NULL.

Files

When compiling a program that uses nvCollectionListCollections, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionListCollections, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionError(3)” on page 103.
- See “nvCollectionFreeList(3)” in “nvCollectionFreeDefn(3)” on page 107.

nvCollectionModify(3)

Purpose

Modifies a collection definition

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int nvCollectionModify(char *name,
                       char *desc,
                       char *rule);
```

Description

The nvCollectionModify function changes the description or rule for an existing collection. Note that this call can use large amounts of CPU time.

Parameters

name
Specifies the name of the collection to be modified

desc
Specifies the modified description for the collection

rule
Specifies the modified rule for determining which objects are included in the collection. Rules can be specified with any of the following:

- A list of IP addresses
- A NetView for AIX field (such as isRouter)
- Another collection name
- A subnet mask or address

Return Values

If successful, nvCollectionModify returns a value of 0. If unsuccessful, nvCollectionModify returns a value of –1.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The specified collection is not defined and cannot be modified.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the modified collection information.

[NV_COLLECTION_INVALID_RULE]
: For some reason, the rule cannot be processed. Check the Boolean logic and any other collection names that you specified.

[NV_COLLECTION_FIELD_NOT_VALID]
: A field specified in the rule to be modified is invalid.

[NV_COLLECTION_PARSING_ERROR]
: The rule cannot be parsed correctly. Check any logical operators you specified.

Files

When compiling a program that uses nvCollectionModify, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionModify, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

nvCollectionOpen(3)

Purpose

Establishes a connection to the collection facility server

Related Functions

nvCollectionXOpen

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int    nvCollectionOpen();

int    nvCollectionXOpen();
```

Description

The nvCollectionOpen function establishes a connection with the collection facility nvcold daemon. If successful, nvCollectionOpen returns a file descriptor. This file descriptor can be used later to register for specific events defined for the collection facility.

nvCollectionXOpen establishes a connection with the collection facility nvcold daemon. Use this routine when you want X Window System to manage your main loop of your program when it is waiting for event notification.

Return Values

If successful, nvCollectionOpen returns a positive value greater than zero. If unsuccessful, nvCollectionOpen returns a negative value.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_ALREADY_INITIALIZED]
A connection to the Collection Facility has already been initiated.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to resolve the collection.

[NV_COLLECTION_CONNECTION_LOST]
The connection to the Collection Facility was interrupted.

Files

When compiling a program that uses nvCollectionOpen, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionOpen, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionAddCallback(3)” on page 98.
- See “nvCollectionError(3)” on page 103.

nvCollectionRead(3)

Purpose

Reads collection facility events

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

int    nvCollectionRead(int sock_fd);
```

Description

The nvCollectionRead function reads the specified socket, and if there is a collection facility event, it calls the callback routine for that event as specified on a nvCollectionAddCallback call. The sock_fd variable specifies the socket on which the response will be received. If there is no data available, nvCollectionRead will not take any action and will return.

This call should be used within a select loop or similar construct. You do not need to use this call if you used nvCollectionXOpen rather than nvCollectionOpen.

Parameters

sock_fd
Specifies the TCP socket to watch for collection facility events.

Files

When compiling a program that uses nvCollectionRead, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionRead, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionDone(3)” on page 102.
- See “nvCollectionOpen(3)” on page 119.
- See “nvCollectionError(3)” on page 103.
- See “nvCollectionErrorMsg(3)” on page 104.

nvCollectionResolve(3)

Purpose

Obtains a list of all objects currently in a specified collection

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

OVwObjectIdList *nvCollectionResolve( char *name);
```

Description

The nvCollectionResolve function returns a list of all objects currently included in the specified collection.

Parameters

name
Specifies the name of the collection

Return Values

If successful, nvCollectionResolve returns a pointer to an OVw object ID list. If unsuccessful, nvCollectionResolve returns NULL.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
The collection specified cannot be found in the list of defined collections.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to resolve the collection.

Files

When compiling a program that uses nvCollectionResolve, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionResolve, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionGetInfo(3)” on page 111.
- See “nvCollectionListCollections(3)” on page 116.

nvCollectionUnion(3)

Purpose

Finds the union of two collections

Related Functions

nvCollectionUnionList

Syntax

```
#include <OV/ovw_obj.h>
#include <OV/OV_nvCollection.h>
#include <OV/OV_nvCollectionErrs.h>

OVwObjectIdList  * nvCollectionUnion(char *name1,
                                     char *name2);

OVwObjectIdList  * nvCollectionListUnion(char *name1,
                                         OVwObjectIdList *list);
```

Description

The nvCollectionUnion function finds the union between two collections. A list is generated of all objects in each of the collections.

nvCollectionListUnion finds the union between a collection and an object list. A list is generated of all objects from the object list as well as those in the collection.

Parameters

name1
Specifies the name of the first collection

name2
Specifies the name of the second collection

list
Specifies a list of object IDs.

Return Values

If successful, nvCollectionUnion and nvCollectionListUnion return a pointer to an OVw object ID list. If unsuccessful, they return NULL.

Error Codes

[NV_COLLECTION_SUCCESS] Successful operation.

[NV_COLLECTION_DOES_NOT_EXIST]
One of the specified collections is not defined.

[NV_COLLECTION_OUT_OF_MEMORY]
There is not enough memory to store the collection information.

Files

When compiling a program that uses nvCollectionUnion or nvCollectionUnionList, you need to include the following files:

- ovw_obj.h
- OV_nvCollection.h
- OV_nvCollectionErrs.h

Libraries

When compiling a program that uses nvCollectionUnion or nvCollectionUnionList, you need to link to the following libraries:

/usr/OV/lib/libov.a

/usr/OV/lib/libovw.a

/usr/OV/lib/libcollection.a

Related Information

- See “nvCollectionIntersect(3)” on page 114.
- See “nvCollectionListIntersect(3)” in “nvCollectionIntersect(3)” on page 114.

nvFilterDefine(3)

Purpose

Creates new filtering rule or updates existing rule

Syntax

```
#include <nvFilter.h>

int nvFilterDefine (struct FilterNode *Filter, char *FileName, char *FilterStr, int Update);
```

Description

The nvFilterDefine function creates a new filtering rule or updates an existing rule. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototypes and the filter structure. The structure has the following definition:

```
struct FilterNode
{
   char *FilterName;
   char *FilterDescription;
   struct FilterNode *Next;
}
```

FilterName	Specifies a pointer to the name of a filtering rule.
FilterDescription	Specifies a pointer to the description of a filtering rule. This field is optional. If no description exists, the pointer to the description is NULL.
Next	Specifies a pointer to the next FilterNode. This field is used only by the nvFilterGetNameList API.

Parameters

Filter	Specifies a pointer to a filter structure containing the name of the filtering rule and optionally a description of the rule.
FileName	Specifies a pointer to the path and name of the filter file.
FilterStr	Specifies a pointer to the content of the filtering rule.
Update	Specifies 0 (zero) for do not update and 1 for update content if rule exists.

Keyword Syntax

! NOT (logical negation)

&& AND (logical and)

|| OR (logical or)

The following list describes the keywords in the syntax used to define filters.

CLASS=value
SNMP enterprise match on enterprise ID. Value is given in dot notation, for example, 1.2.3.4.55

IP_ADDR=value
SNMP agent-addr match on IP address. Value is given in dot notation, for example, 192.155.13.57. Registration for an IP_ADDR permits receipt of agent-generated traps, as well as internal events related to that IP_ADDR.

LOGGED_TIME <= time_string
Time that was logged before the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

LOGGED_TIME >= time_string
Time that was logged after the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

PRESENT = SNMP_TRAP
Presence of SNMP Trap

SNMP_TRAP=value
Match on SNMP Generic Trap Type, where the Generic Type is an integer

SNMP_SPECIFIC=value
Match on SNMP Specific Trap Type, where the Specific Type is an integer

TIME_PERIOD=time_constant
Relative time period (integer seconds) for frequency filters

THRESHOLD <= frequency
Number of event occurrences is less than or equal to frequency (integer) during TIME_PERIOD

THRESHOLD >= frequency
Number of event occurrences is greater than or equal to frequency (integer) during TIME_PERIOD

Note: When included in an expression for nvSnmpTrapOpenFilter, the keywords THRESHOLD and TIME_PERIOD must be ANDed (never ORed) and grouped within parentheses as in the following example:

```
filter = PRESENT=SNMP_TRAP && (THRESHOLD <= 5 && TIME_PERIOD = 30)
```

Specifying more than 250 filter objects will result in an error.

Return Values

If successful, nvFilterDefine returns 0 (zero). If unsuccessful, it returns one of the following nonzero error codes.

Error Codes

[NVFILTER_FILE_ACCESS_ERROR] The filter file could not be accessed. Check the file permissions and try again.

[NVFILTER_MEMORY_ACCESS_ERROR]
Memory could not be allocated.

[NVFILTER_DUPLICATE_FILTERNAME] The filter name already exists in the file and the caller did not specify an update.

[NVFILTER_MAX_BUFFERSIZE_EXCEEDED]
The filter contents, the filter description, plus the keywords exceeded 20K bytes, which is the maximum buffer size.

Libraries

When you are compiling a program that uses nvFilterDefine, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDelete(3)” on page 129.
- See “nvFilterErrorMsg(3)” on page 130.
- See “nvFilterFreeNameList(3)” on page 131.
- See “nvFilterGet(3)” on page 132.
- See “nvFilterGetNameList(3)” on page 134.

nvFilterDelete(3)

Purpose

Removes a filtering rule from the filter file

Syntax

```
#include <nvFilter.h>

int nvFilterDelete(char *RuleName, char *FileName);
```

Description

The nvFilterDelete function removes a filtering rule from the filter file. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototype.

Parameters

RuleName	Specifies a pointer to the name of the filtering rule to delete.
FileName	Specifies a pointer to the path and name of the filter file.

Return Values

If successful, nvFilterDelete returns 0 (zero). If unsuccessful, it returns one of the following nonzero error codes.

Error Codes

[NVFILTER_FILE_NOT_FOUND]	The specified filter file was not found.
[NVFILTER_FILE_ACCESS_ERROR]	The filter file could not be accessed. Check the file permissions and try again.
[NVFILTER_MEMORY_ACCESS_ERROR]	Memory could not be allocated.
[NVFILTER_FILTERNAME_NOT_FOUND]	The specified filtername was not found in the file.

Libraries

When compiling a program that uses nvFilterDelete, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDefine(3)” on page 126.
- See “nvFilterErrorMsg(3)” on page 130.
- See “nvFilterFreeNameList(3)” on page 131.
- See “nvFilterGet(3)” on page 132.
- See “nvFilterGetNameList(3)” on page 134.

nvFilterErrorMsg(3)

Purpose

Retrieves the error message that corresponds to an nvFilter API return code

Syntax

```
#include <nvFilter.h>
char *nvFilterErrorMsg(int Retcode);
```

Description

The nvFilterErrorMsg function retrieves the error message that corresponds to an nvFilter API return code. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototype.

Parameters

Retcode Specifies the return code from an unsuccessful nvFilter API call.

Return Values

If successful, nvFilterErrorMsg returns the error message. If unsuccessful, it returns a NULL string.

Libraries

When compiling a program that uses nvFilterErrorMsg, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDefine(3)” on page 126.
- See “nvFilterDelete(3)” on page 129.
- See “nvFilterFreeNameList(3)” on page 131.
- See “nvFilterGet(3)” on page 132.
- See “nvFilterGetNameList(3)” on page 134.

nvFilterFreeNameList(3)

Purpose

Frees the memory allocated during the creation of a list of filtering rule names

Syntax

```
#include <nvFilter.h>

void nvFilterFreeNameList (struct FilterNode *FilterList);
```

Description

The nvFilterFreeNameList function frees the memory allocated during the creation of the list of filtering rule names. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototypes and the filter structure. The structure has the following definition:

```
struct FilterNode
{
   char *FilterName;
   char *FilterDescription;
   struct FilterNode *Next;
}
```

FilterName	Specifies a pointer to the name of a filtering rule.
FilterDescription	Specifies a pointer to the description of a filtering rule. The field is optional. If no description exists, the pointer to the description is NULL.
Next	Specifies a pointer to the next FilterNode. This field is used only by the nvFilterGetNameList API.

Parameters

FilterList	Specifies a pointer to the first item in the filter list.
Return Values	There is no return parameter.

Libraries

When compiling a program that uses nvFilterFreeNameList, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDefine(3)” on page 126.
- See “nvFilterDelete(3)” on page 129.
- See “nvFilterErrorMsg(3)” on page 130.
- See “nvFilterGet(3)” on page 132.
- See “nvFilterGetNameList(3)” on page 134.

nvFilterGet(3)

Purpose

Retrieves the contents of the filtering rule

Syntax

```
#include <nvFilter.h>

int nvFilterGet (struct FilterNode *Filter, char *FileName,
       char *Buffer, int *BufLen, int Expand);
```

Description

The nvFilterGet function retrieves the contents of the filtering rule. This function returns the first occurrence of the filter name. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototypes and the filter structure. The structure has the following definition:

```
struct FilterNode
{
   char *FilterName;
   char *FilterDescription;
   struct FilterNode *Next;
}
```

FilterName	Specifies a pointer to the name of a filtering rule.
FilterDescription	Specifies a pointer to the description of a filtering rule. This field is optional. If no description exists, the pointer to the desciption is NULL.
Next	Specifies a pointer to the next FilterNode. This field is used only by the nvFilterGetNameList API.

Space is allocated for the rule description on each call to the nvFilterGet routine. Your application is responsible for deallocating this memory between calls to this routine. This technique is illustrated in *NetView for AIX Programmer's Guide*.

Parameters

Filter	Specifies a pointer to a filter structure containing the name of the filtering rule. If a description exists for the specified rule, it is returned in the FilterDescription field.
FileName	Specifies a pointer to the path and name of the filter file.
Buffer	Specifies a pointer to the memory location where the rule will be written. If this field is NULL, an error is returned and the length of the filtering rule will be returned in the BufLen parameter.
BufLen	Specifies a pointer to the size of the buffer. If the buffer is too small, an error is returned and BufLen is changed to reflect the actual size of the filtering rule.
Expand	Specifies 0 (zero) for do not expand references to other filtering rules and hostnames, and 1 for expand the references.

Return Values

If successful, nvFilterGet returns 0 (zero), unless the buffer field is NULL. If so, nvFilterGet returns [NVFILTER_INSUFFICIENT_SPACE]. If unsuccessful, it returns one of the following nonzero error codes.

Error Codes

[NVFILTER_FILE_NOT_FOUND] The specified filter file was not found.

[NVFILTER_FILE_ACCESS_ERROR]
The filter file could not be accessed. Check the file permissions and try again.

[NVFILTER_MEMORY_ACCESS_ERROR]
Memory could not be allocated.

[NVFILTER_INSUFFICIENT_SPACE]
The buffer was too small for the specified filtering rule. BufLen is set to the size of the rule.

[NVFILTER_FILTERNAME_NOT_FOUND]
The specified filtername was not found in the file.

[NVFILTER_HOSTNAME_RESOLUTION_ERROR]
The reference to a hostname could not be resolved.

[NVFILTER_FILTER_RESOLUTION_ERROR]
The reference to another filtering rule could not be expanded.

[NVFILTER_FILTER_REFERENCE_ERROR]
The filtering rule has an incorrect format.

[NVFILTER_TIME_FORMAT_ERROR]
The time has an incorrect format. It must be in the format HH:MM:SS.

Libraries

When compiling a program that uses nvFilterGet, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDefine(3)” on page 126.
- See “nvFilterDelete(3)” on page 129.
- See “nvFilterErrorMsg(3)” on page 130.
- See “nvFilterFreeNameList(3)” on page 131.
- See “nvFilterGetNameList(3)” on page 134.

nvFilterGetNameList(3)

Purpose

Retrieves a list of all filtering rules in a filter file

Syntax

```
#include <nvFilter.h>

int nvFilterGetNameList (char *FileName, struct FilterNode **FilterList);
```

Description

The nvFilterGetNameList function retrieves a list of all the unique filtering rule names in the filter file. If the filter file contains duplicate names, the nvFilterGetNameList function does not return these duplicates. An include file, /usr/OV/include/nvFilter.h, is provided. This file contains the function prototypes and the filter structure. The structure has the following definition:

```
struct FilterNode
{
   char *FilterName;
   char *FilterDescription;
   struct FilterNode *Next;
}
```

FilterName	Specifies a pointer to the name of a filtering rule.
FilterDescription	Specifies a pointer to the description of a filtering rule. This field is optional. If no description exists, the pointer to the description is NULL.
Next	Specifies a pointer to the next FilterNode. This field is used only by the nvFilterGetNameList API.

Parameters

FileName	Specifies the path and name of the filter file.
FilterList	Specifies the address of a pointer to be set by nvFilterGetNameList. If the function returns successfully, it will contain the address of the first FilterNode in the linked list.

Return Values

If successful, nvFilterGetNameList returns 0 (zero). If unsuccessful, it returns one of the following nonzero error codes.

Error Codes

[NVFILTER_FILE_NOT_FOUND]	The filter file was not found.
[NVFILTER_FILE_ACCESS_ERROR]	The filter file could not be accessed. Check the file permissions and try again.
[NVFILTER_MEMORY_ACCESS_ERROR]	Memory could not be allocated.

[NVFILTER_FILTER_FILE_EMPTY] The specified filter file is empty.

[NVFILTER_INCORRECT_FILTER_FILE_FORMAT]
The specified filter file did not contain any filtering rules that were in the correct format.

Libraries

When compiling a program that uses nvFilterGetNameList, you need to link to the following library:

/usr/OV/lib/libnvfilter.a

Related Information

- See “nvFilterDefine(3)” on page 126.
- See “nvFilterDelete(3)” on page 129.
- See “nvFilterErrorMsg(3)” on page 130.
- See “nvFilterFreeNameList(3)” on page 131.
- See “nvFilterGet(3)” on page 132.

NVisClient(3)

Purpose

Checks to see if an application is running on a client or a server

Syntax

```
#include <nvDefServ.h>

int NVisClient ();
```

Description

The NVisClient function determines whether an application is running on a NetView for AIX client or server, for example, a distributed application that has daemons on both the server and clients.

Applications that are launched from the menu bar and are installed on each client probably do not need to know if they are running on a client or server.

Return Values

If the machine is a server, NVisClient returns a value of 0 (FALSE). If the machine is a client, NVisClient returns a value of 1 (TRUE).

Files

When compiling a program that uses NVisClient, you need to include the following files:

- nvDefServ.h

Libraries

When compiling a program that uses NVisClient, you need to link to the following library:

/usr/OV/lib/libovw.a

Related Information

- See “OVDefaultServerName(3)” on page 424.

nvotChangeArcDetails(3)

Purpose

Changes the contents of the details variable in the database

Syntax

```
nvotReturnCode     nvotChangeArcDetails (
                    nvotNameBindingType    arcNameBinding,
                    nvotProtocolType       aEndpointProtocol,
                    char              *    aEndpointName,
                    nvotProtocolType       zEndpointProtocol,
                    char              *    zEndpointName,
                    int                    arcIndexId,
                    nvotOctetString   *    arcDetails)
```

Description

The nvotChangeArcDetails routine changes the contents of the details variable associated with the arc named by aEndpoint, zEndpoint and arcIndexId.

An arc connects arc endpoints: two vertices, two graphs, a vertex to a graph or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId.

The arcNameBinding parameter helps to identify the arc endpoints. See the following parameters section for a detailed description. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of class graph must exist; otherwise, the arc details are not changed and the error codes NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST or NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST are set. The GTM interface does not support automatic creation of graphs.

If an endpoint of class vertex does not exist, it is automatically created. Also, the arc is created with default values and the details changed. This is part of GTM's recovery strategy for lost traps. However, a vertex endpoint is NOT created if the other endpoint is a reference to a nonexistent graph.

The nvotProtocolType is a union of an enumerated type with a char pointer as defined in the nvotTypes.h file. Special care must be taken when setting aEndpointProtocol and zEndpointProtocol. Setting these variables to a nvotVertexProtocolType value if arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

Parameters

arcNameBinding Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

ARC_VERTEX_VERTEX_NAME_BINDING
: Indicates that either endpoint is a vertex

ARC_VERTEX_GRAPH_NAME_BINDING
: Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
Indicates that aEndpoint is a graph and zEndpoint is a vertex

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates that either endpoint is a graph

If any value other than the preceding values is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs can be handled based on their direction. For more information about the direction of arcs, see “nvotInit(3)” on page 359. Regardless of which direction was set in the nvotInit routine, the arcNameBinding parameter always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. If aEndpoint or zEndpoint is to be a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set with a value from the enumerated type nvotVertexProtocolType, which is defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName/zEndPointName
Specifies the name of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. Both the endpoint name and the endpoint protocol are required to identify the object at one of the endpoints of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

arcDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(arcDetails->octetString, (char *) applStruct, sizeof(applStruct)) and arcDetails->octetLength = sizeof(applStruct). Although nvotOctetString allows for any size strings and the interface does not check the size of boxDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

Return Values

nvotReturnCode The nvotChangeArcDetails routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST] The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST] The graph defined as the Z endpoint of the arc does not exist in the GTM database.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the character string **mystring** in the *myLineArcDetails* variable of the arc created in the example in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;
char  myString [50] = {"The quick brown fox jumped over the lazy dogs back"};

/*************** Define vertices V1 and V2 *****************/
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;
char         *         myLineArcLabel = "My_Line_Arc";
int                    arcNumber = 1;

nvotOctetString    *   myLineArcDetails = NULL;

myLineArcDetails.octetString = malloc (sizeof (myString));
myLineArcDetails.octetLength = (sizeof (myString));
memcpy (myLineArcDetails.octetString, :myString, sizeof (myString));

  if ((rc = nvotChangeArcDetails (ARC_VERTEX_VERTEX_NAME_BINDING,
                                  oneEndpointlType,
```

```
                                oneEndpointName,
                                otherEndpoint,
                                otherEndpointName,
                                arcNumber,
                                myLineArcDetails)) == NVOT_SUCCESS)

  printf ("myString has been stored in %s.\n", myLineArcLabel);
else
  printf ("Error occurred storing myString in %s.\n", myLineArcLabel);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.

nvotChangeArcIconInGraph(3)

Purpose

Changes an arc icon in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeArcIconInGraph (
                        nvotGraphProtocolType graphProtocol,
                        char      *           graphName,
                        nvotNameBindingType   arcNameBinding,
                        nvotProtocolType      aEndpointProtocol,
                        char      *           aEndpointName,
                        nvotProtocolType      zEndpointProtocol,
                        char      *           zEndpointName,
                        int                   arcIndexId,
                        char      *           icon)
```

Description

The nvotChangeArcIconInGraph routine changes the icon representing the arc identified by aEndpointProtocol, zEndpointName, zEndpointProtocol, zEndpointName, and arcIndexId that is associated with the graph identified by graphProtocol and graphName.

The containing graph must exist. Otherwise, the arc icon is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

An arc connects arc endpoints: two vertices, two graphs, a vertex to a graph, or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId. The following parameters are required:

- graphProtocol
- graphName
- aEndpointProtocol
- aEndpointName
- zEndpointProtocol
- zEndpointName
- arcIndexId

If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_ARC_INVALID_INDEX is returned.

The arcNameBinding parameter helps to identify the arc endpoints. See the parameters section for a detailed description. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of class graph must exist. Otherwise the arc icon is not changed and either the error code NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST or NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST is returned.

If endpoints of class vertex do not exist, they are automatically created and the arc icon is changed. This is part of the GTM's recovery strategy for lost traps. However, a vertex endpoint is NOT created and the arc icon is not changed if the other endpoint is a reference to a nonexistent graph.

The nvotProtocolType is a union of an enumerated type with a char pointer as defined in the nvotTypes.h file. Special care must be taken when setting the aEndpointProtocol and zEndpointProtocol parameters. To set these variables with an nvotVertexProtocolType value if arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

To be supported by the nvotChangeArcIconInGraph routine, the icon must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if the icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the arc icon is changed to the default symbol **Connection:Generic**.

Parameters

graphProtocol
Specifies the protocol of the graph that contains the arc. This is the graph of which this arc is a member arc. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName
Specifies the name of the graph that contains the arc. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter is a string of characters used to create the graph.

arcNameBinding
Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates that either endpoint is a vertex.

ARC_VERTEX_GRAPH_NAME_BINDING
Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
Indicates that aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either endpoint is a graph

If a value other than those in the preceding list is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs are handled based on their direction. For more information about arc direction, see “nvotInit(3)” on page 359. Regardless of the selection made in the nvotInit routine, arcNameBinding always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or aEndpointProtocol, respectively, of this arc. If aEndpoint or zEndpoint is a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set to a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName Specifies the name of the object identified as the aEndpoint of this arc. Both the aEndpointName and aEndpointProtocol parameters are required to identify the object at the aEndpoint of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

icon Specifies a new symbol to represent the arc in the NetView for AIX external user interface. The symbol can be a line, a dotted line, and so forth. For details about selecting an icon, refer to the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeArcIconInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX] The endpoint graph index is not valid. An endpoint graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST] The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST] The graph defined as the Z endpoint of the arc does not exist in the GTM database.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the arc created in the example in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;

/*************** Define the parent graph *********************/
nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Graph";

/*************** Define vertices V1 and V2 *****************/
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;

/*************** Define arcs attributes ****************************/
char         *         myDotDashArcIcon = "1.3.6.1.2.1.2.2.1.3.53.4";

char         *         myLineArcLabel = "My_Line_Arc"
int                    arcNumber = 1;

  if ((rc = nvotChangeArcIconInGraph (myGraphProt,
                                      myGraphName,
                                      ARC_VERTEX_VERTEX_NAME_BINDING,
                                      oneEndpoint,
                                      oneEndpointName,
                                      otherEndpoint,
                                      otherEndpointName,
                                      arcNumber,
                                      myDotDashArcIcon)) == NVOT_SUCCESS)

    printf ("Arc icon of %s changed.\n", myLineArcLabel);
  else
    printf ("An error occurred changing %s icon.\n", myLineArcLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotChangeArcLabelInGraph(3)” on page 146.
- See “nvotCreateArcInGraph(3)” on page 221.

nvotChangeArcLabelInGraph(3)

Purpose

Changes an arc label in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeArcLabelInGraph (
                        nvotGraphProtocolType graphProtocol,
                        char      *           graphName,
                        nvotNameBindingType   arcNameBinding,
                        nvotProtocolType      aEndpointProtocol,
                        char      *           aEndpointName,
                        nvotProtocolType      zEndpointProtocol,
                        char      *           zEndpointName,
                        int                   arcIndexId,
                        char      *           label)
```

Description

The nvotChangeArcLabelInGraph routine changes the label of the arc identified by aEndpointProtocol, zEndpointName, zEndpointProtocol, zEndpointName, and arcIndexId that is associated with the graph identified by graphProtocol and graphName.

The containing graph must exist. Otherwise, the arc label is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

An arc connects arc endpoints: two vertices, two graphs, a vertex to a graph, or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId. The following parameters are required:

- graphProtocol
- graphName
- aEndpointProtocol
- aEndpointName
- zEndpointProtocol
- zEndpointName
- arcIndexId

If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_ARC_INVALID_INDEX is returned.

The arcNameBinding parameter helps to identify the arc endpoints. See the parameters section for a detailed description. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of class graph must exist. Otherwise the arc icon is not changed and either the error code NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST or NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST is returned.

If endpoints of class vertex do not exist, they are automatically created and the arc icon is changed. This is part of the GTM's recovery strategy for lost traps. However, a vertex endpoint is NOT created and the arc icon is not changed if the other endpoint is a reference to a nonexistent graph.

The nvotProtocolType is a union of an enumerated type with a char pointer as defined in the nvotTypes.h file. Special care must be taken when setting the aEndpointProtocol and zEndpointProtocol parameters. To set these variables with an nvotVertexProtocolType value if arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

The label parameter is a character string displayed under a symbol in the NetView for AIX EUI. Usually, it is a human-readable character string that helps to visually identify a network resource. Although the label must be a valid pointer, NULL is accepted. A pointer that is not valid can cause unpredictable errors. If NULL is passed, a concatenation of aEndpointName + zEndpointName + arcIndexId is displayed in place of the label.

Parameters

graphProtocol
Specifies the protocol of the graph that contains the arc. This is the graph of which this arc is a member arc. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName
Specifies the name of the graph that contains the arc. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter is a string of characters used to create the graph.

arcNameBinding
Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates that either endpoint is a vertex.

ARC_VERTEX_GRAPH_NAME_BINDING
Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
Indicates that aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either endpoint is a graph

If a value other than those in the preceding list is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs are handled based on their direction. For more information about arc direction, see “nvotInit(3)” on page 359. Regardless of the selection made in the nvotInit routine, arcNameBinding always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or aEndpointProtocol, respectively, of this arc. If aEndpoint or zEndpoint is a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set to a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName Specifies the name of the object identified as the aEndpoint of this arc. Both the aEndpointName and aEndpointProtocol parameters are required to identify the object at the aEndpoint of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

label When you click the right mouse button on an arc symbol, a pull-down menu is displayed. In its upper line, the menu shows a label for the arc. The arc label parameter specifies a string of characters to be displayed in this pull-down menu.

Return Values

nvotReturnCode The nvotChangeArcLabelInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the Z endpoint of the arc does not exist in the GTM database.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the arc created in the example in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;

/*************** Define the parent graph *********************/
nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Graph";

/*************** Define vertices V1 and V2 *****************/
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;

/*************** Define arcs attributes ****************************/
char         *         myDotDashArcLabel = "My_Dotted_Arc"
char         *         myLineArcLabel = "My_Line_Arc"
int                    arcNumber = 1;

  if ((rc = nvotChangeArcLabelInGraph (myGraphProt,
                                       myGraphName,
                                       ARC_VERTEX_VERTEX_NAME_BINDING,
                                       oneEndpoint,
                                       oneEndpointName,
                                       otherEndpoint,
                                       otherEndpointName,
                                       arcNumber,
                                       myDotDashArcLabel)) == NVOT_SUCCESS)

    printf ("Arc label of %s changed.\n", myLineArcLabel);
  else
    printf ("An error occurred changing %s label.\n", myLineArcLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotChangeArcIconInGraph(3)” on page 141.
- See “nvotCreateArcInGraph(3)” on page 221.

nvotChangeArcStatus(3)

Purpose

Changes the status of an arc

Syntax

```
nvotReturnCode     nvotChangeArcStatus (
                    nvotNameBindingType    arcNameBinding,
                    nvotProtocolType       aEndpointProtocol,
                    char *                 aEndpointName,
                    nvotProtocolType       zEndpointProtocol,
                    char *                 zEndpointName,
                    int                    arcIndexId,
                    statusType             arcStatus)
```

Description

The nvotChangeArcStatus routine changes the status of an arc named by aEndpoint, zEndpoint and arcIndexId.

An arc connects arc endpoints: two vertices, two graphs, a vertex to a graph, or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId.

The arcNameBinding parameter helps to identify the arc endpoints. See the following parameters section for a detailed description. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of class graph must exist; otherwise the arc status is not changed and the error codes NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST or NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST is set. The GTM interface does not support automatic creation of graphs.

If an endpoint of class vertex does not exist, it is automatically created. Also, the arc is created with default values and the status changed. This is part of the GTM's recovery strategy for lost traps. However, a vertex endpoint is NOT created if the other endpoint is a reference to an nonexistent graph.

The nvotProtocolType is a union of an enumerated type with a char pointer as defined in the nvotTypes.h file. Special care must be taken when setting aEndpointProtocol and zEndpointProtocol. Setting these variables to a nvotVertexProtocolType value if arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

The arcStatus parameter reflects the status of a network connection. The statusType is defined in the file nvotTypes.h. The possible values are mapped into a combination of four status attributes: operational state, alarm status, availability status, and unknown status. For a detailed explanation, see the section about state management variables in the *NetView for AIX Programmer's Guide*. If the value passed is not valid, the operation is rejected and error code NVOT_INVALID_STATUS is returned.

Parameters

arcNameBinding

Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

ARC_VERTEX_VERTEX_NAME_BINDING
: Indicates that either endpoint is a vertex

ARC_VERTEX_GRAPH_NAME_BINDING
: Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
: Indicates that aEndpoint is a graph and zEndpoint is a vertex

ARC_GRAPH_GRAPH_NAME_BINDING
: Indicates that either endpoint is a graph

If any value other than the preceding values is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs can be handled based on their direction. For more information about the direction of arcs, see “nvotInit(3)” on page 359. Regardless of which direction was set in the nvotInit routine, the arcNameBinding parameter always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol

Specifies the protocol of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. If aEndpoint or zEndpoint is to be a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set with a value from the enumerated type nvotVertexProtocolType, which is defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName/zEndPointName

Specifies the name of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. Both the endpoint name and the endpoint protocol are required to identify the object at one of the endpoints of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId

Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

arcStatus

Specifies a set of values to represent the status of a connection. This parameter is a combination of MIB variables OperationalState, AlarmStatus, AvailabilityStatus and UnknownStatus. The statusType is defined in the file nvotTypes.h.

Return Values

nvotReturnCode

The nvotChangeArcStatus routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]	The endpoint graph index is not valid. An endpoint graph protocol or name must not be NULL.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. It must be a positive integer.
[NVOT_INVALID_STATUS]	The status is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The graph defined as the A endpoint of the arc does not exist in the GTM database.
[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The graph defined as the Z endpoint of the arc does not exist in the GTM database.
[NVOT_INVALID_NAME_BINDING]	The name binding is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the status of one the arcs created in the example given in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;
char         *         myLineArcLabel = "My_Line_Arc";
int                    arcNumber = 1;

nvotStatusType         myLineArcStatus = STATUS_MARGINAL;

  if ((rc = nvotChangeArcStatus (ARC_VERTEX_VERTEX_NAME_BINDING,
```

```
                                oneEndpoint,
                                oneEndpointName,
                                otherEndpoint,
                                otherEndpointName,
                                arcNumber,
                                myLineArcStatus)) == NVOT_SUCCESS)

  printf ("Arc status of arc %s changed.\n", myLineArcLabel);
else
  printf ("An error occurred changing %s status.\n", myLineArcLabel);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.
- See “nvotGetArcsInGraph(3)” on page 307.
- See “nvotInit(3)” on page 359.

nvotChangeBoxBackground(3)

Purpose

Changes the background of a box map

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeBoxBackground (
                        nvotGraphProtocolType  boxProtocol,
                        char         *         boxName,
                        char         *         boxBackground)
```

Description

The nvotChangeBoxBackground routine changes the image displayed in the background of the submap into which the box given by boxProtocol and boxName is exploded.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol and boxName parameters are required. If one of these parameters is not provided, the error code NVOT_BOX_INVALID_INDEX is returned.

If the graph specified does not exist in the GTM database, its submap does not exist either and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

If a graph that matches boxProtocol and boxName but whose graphType attribute is not set to BOX exists in the GTM database, its background is not changed and either NVOT_GRAPH_ALREADY_EXIST or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

Parameters

boxProtocol Specifies the protocol of the child box graph. For more information about specifying a box graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the child box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the child box graph. This parameter is a string of characters used to create the box graph.

boxBackground Specifies an image to be displayed in the background of the submap into which this box is exploded. Background is usually an image of a geographic region that helps to illustrate a submap. You can select a background image from among the bitmap files in the default directory **/usr/OV/backgrounds**.

Return Values

nvotReturnCode The nvotChangeBoxBackground routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box graph protocol or name must not be NULL.
[NVOT_BOX_DOES_NOT_EXIST]	The box graph does not exist in the GTM database.
[NVOT_GRAPH_ALREADY_EXIST]	A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.
[NVOT_OTHER_TYPE_GRAPH_EXIST]	Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the background in the submap of the graph created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";

char         *         my_NEW_BackgroundMap  = "usa";

  if ((rc = nvotChangeBoxBackground (my_STARLAN_GraphsProt,
                                     myBox_STARLAN_GraphName,
                                     my_NEW_BackgroundMap)) == NVOT_SUCCESS)

    printf ("Background of box graph %s changed.\n", myBox_STARLAN_GraphName);
  else
    printf ("Error occurred changing %s background.\n",myBox_STARLAN_GraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotCreateBoxInGraph(3)” on page 227.

nvotChangeBoxDetails(3)

Purpose

Changes the contents of the details variable in the GTM database

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeBoxDetails (
                         nvotGraphProtocolType  boxProtocol,
                         char              *    boxName,
                         nvotOctetString   *    boxDetails)
```

Description

The nvotChangeBoxDetails routine changes the contents of the details variable associated with the box graph identified by boxProtocol and boxName.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol and boxName parameters are required. If one of these parameters is not provided, the error code NVOT_BOX_INVALID_INDEX is returned.

If the box graph specified does not exist in the GTM database, the operation is rejected and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

If a graph that matches boxProtocol and boxName but whose graphType attribute is not set to BOX exists in GTM database, its details variable is not changed and either NVOT_GRAPH_ALREADY_EXIST or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

Parameters

boxProtocol Specifies the protocol of the box graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the box graph in the GTM database. This parameter is a string of characters used to create the box graph.

boxDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(boxDetails->octetString, (char *) applStruct, sizeof(applStruct)) and boxDetails->octetLength = sizeof(applStruct). Although nvotOctetString allows for any size strings and the interface does not check the size of boxDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

Return Values

nvotReturnCode The nvotChangeBoxDetails routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box graph protocol or name must not be NULL.
[NVOT_BOX_DOES_NOT_EXIST]	The box graph does not exist in the GTM database.
[NVOT_GRAPH_ALREADY_EXIST]	A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.
[NVOT_OTHER_TYPE_GRAPH_EXIST]	Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the contents of **myStruct** snd **myString** into the **boxDetails** variable attribute of My_Box.

```
#include <nvot.h>
typedef struct { int var1;
                 int var2;
               } structType;

structType myStruct = { 11, 22 };
char  myString [50] = ["The quick brown fox jumped over the lazy dogs back"];

nvotOctetString       myBoxDetails;
char        *         auxDetailsPtr;
myBoxDetails.octetString = malloc (sizeof (myStruct) + sizeof (myString));
myBoxDetails.octetLength = (sizeof (structType) + sizeof (myString));
auxDetailsPtr = myBoxDetails.octetString;

memcpy (myBoxDetails.octetString, (char *) :myStruct, sizeof (myStruct));
auxDetailsPtr = myBoxDetails.octetString + sizeof (myStruct);
memcpy (auxDetailsPtr, :myString, sizeof (myString));

nvotReturnCode         rc;
```

```
nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myBoxName = "My_Box";

  if ((rc = nvotChangeBoxDetails (myBoxProt,
                                  myBoxName,
                                  :myBoxDetails)) == NVOT_SUCCESS)

    printf ("myString has been stored in %s.\n", myBoxName);
  else
    printf ("Error occurred storing myString in %s.\n", myBoxName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateBoxInGraph(3)” on page 227.

nvotChangeBoxIconInGraph(3)

Purpose

Changes a box graph icon in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode  nvotChangeBoxIconInGraph (
                        nvotGraphProtocolType  graphProtocolParent,
                        char                   graphNameParent,
                        nvotGraphProtocolType  boxProtocol,
                        char             *     boxName,
                        char             *     icon)
```

Description

The nvotChangeBoxIconInGraph routine changes the icon representing the box graph identified by boxProtocol and boxName and that is displayed in the submap of the graph identified by graphProtocolParent and graphNameParent.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, boxProtocol and boxName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If either the parent or box graph does not exist in the GTM database, the box graph icon is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST or NVOT_BOX_DOES_NOT_EXIST is returned. Automatic creation of box graphs is not supported.

To be supported by the nvotChangeBoxIconInGraph routine, the icon must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if the icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the box graph icon is changed to the default symbol **Computer:Generic**.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in GTM the database. This parameter is a string of characters used to create the parent graph.

boxProtocol Specifies the protocol of the child box graph. For more information about specifying a box graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the child box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the child box graph. This parameter is a string of characters used to create the box graph.

icon Specifies a symbol to represent the child box graph in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeBoxIconInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol or name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	A graph does not exist in the GTM database.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box graph protocol and name must not be NULL.
[NVOT_BOX_DOES_NOT_EXIST]	The box graph does not exist in the GTM database.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the box created in the example in “nvotCreateBoxInGraph(3)” on page 227.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";

char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";
char         *         my_NEW_BoxIcon = "1.3.6.1.2.1.2.2.1.3.9.10";

  if (rc = nvotChangeBoxIconInGraph (my_STARLAN_GraphsProt,
                                     myRoot_STARLAN_GraphName,
                                     my_STARLAN_GraphsProt,
                                     myBox_STARLAN_GraphName,
                                     my_NEW_BoxIcon) == NVOT_SUCCESS)
```

```
  printf ("Box icon of box graph %s changed.\n", myBox_STARLAN_GraphName);
else
  printf ("An error occurred changing %s icon.\n", myBox_STARLAN_GraphName);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotChangeBoxLabelInGraph(3)” on page 163.

nvotChangeBoxLabelInGraph(3)

Purpose

Changes a box graph label in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeBoxLabelInGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  boxProtocol,
                         char         *         boxName,
                         char         *         label)
```

Description

The nvotChangeBoxLabelInGraph routine changes the label of the box graph identified by boxProtocol and boxName and displayed in the submap of the graph identified by graphProtocolParent and graphNameParent.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, boxProtocol and boxName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If either the parent or child box graph does not exist in the GTM database, the box label is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST or NVOT_BOX_DOES_NOT_EXIST is returned. Automatic creation of box graphs is not supported.

The label parameter is a character string displayed under a symbol in the NetView for AIX EUI. Usually, it is a human-readable character string that helps to visually identify a resource in a topology map. Although the label must be a valid pointer, NULL is accepted. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the boxName string is displayed in place of the label.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in GTM the database. This parameter is a string of characters used to create the parent graph.

boxProtocol Specifies the protocol of the child box graph. For more information about specifying a box graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the child box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the child box graph. This parameter is a string of characters used to create the box graph.

label Specifies a human-readable character string to be displayed under the box graph symbol in the NetView for AIX EUI. This parameter must be a valid character string or NULL.

Return Values

nvotReturnCode The nvotChangeBoxLabelInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol or name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the box graph created in the example in “nvotCreateBoxInGraph(3)” on page 227.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";

char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";
char         *         my_NEW_BoxLabel = "My_Workstation_Box";
```

```
if (rc = nvotChangeBoxLabelInGraph (my_STARLAN_GraphsProt,
                                    myRoot_STARLAN_GraphName,
                                    my_STARLAN_GraphsProt,
                                    myBox_STARLAN_GraphName,
                                    my_NEW_BoxLabel) == NVOT_SUCCESS)

  printf ("Box icon of box graph %s changed.\n", myBox_STARLAN_GraphName);
else
  printf ("An error occurred changing %s icon.\n", myBox_STARLAN_GraphName);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotChangeBoxIconInGraph(3)” on page 160.

nvotChangeBoxPositionInGraph(3)

Purpose

Changes the position of a box graph icon in a graph submap

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeBoxPositionInGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  boxProtocol,
                         char         *         boxName,
                         nvotPositionType       newPosition)
```

Description

The nvotChangeBoxPositionInGraph routine changes the position of a symbol representing the box graph identified by boxProtocol and boxName and associated with the graph identified by graphProtocolParent and graphNameParent.

The parent graph must have been created with the layout algorithm set to NONE_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, boxProtocol, and boxName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the parent graph does not exist or it exists but its graphType is not set to GRAPH, the box position is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If the box graph does not exist or it exists but its graphType is not set to BOX, the box position is not changed and the error code NVOT_BOX_DOES_NOT_EXIST is returned. Automatic creation of graph is not supported.

The nvotPositionType parameter, as defined in the file nvotTypes.h, accepts the following four variables: xCoordinate, yCoordinate, xGrid and yGrid. The xGrid and yGrid variables determine a scale on which the coordinate system is defined.

The grid and coordinate do not necessarily determine the exact physical location in the window where the symbol is displayed. However, they determine a virtual position for the symbol based on the virtual size of the submap.

The symbol size can be affected either by the grid values or by the coordinate values. For example, if the symbol position is set too far from the center or from another symbol, x and y grid are reset to a value that keeps the distances proportional and allows all symbols in the submap to be displayed. This placement of symbolx has the effect of a zoomout.

For best results, use the same xGrid and yGrid values for all symbols in the same submap.

Parameters

graphProtocolParent Specifies the protocol of the parent graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph.

boxProtocol Specifies the protocol of the box graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the box graph. This parameter is a string of characters used to create the box graph.

newPosition Specifies the values of the variables xCoordinate, yCoordinate, xGrid and yGrid in a structure defined in the file nvotTypes.h.

Return Values

nvotReturnCode The nvotChangeBoxPositionInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol or name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the position of a child box graph.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";
char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";

nvotPositionType       myBoxGraphPosition = { 500,  /* xCoordinate */
                                              500,  /* yCoordinate */
                                              1000, /* xGrid       */
                                              1000  /* yGrid       */
                                            };

  if ((rc = nvotChangeBoxPositionInGraph (my_STARLAN_GraphsProt,
                                          myRoot_STARLAN_GraphName,
                                          my_STARLAN_GraphsProt,
                                          myBox_STARLAN_GraphName,
                                          myBoxGraphPosition)) == NVOT_SUCCESS)

    printf ("Positioning of graph %s symbol changed.\n",
                                                    myBox_STARLAN_GraphName);
  else
    printf ("An error occurred changing %s icon position.\n",
                                                    myBox_STARLAN_GraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotChangeGraphPositionInGraph(3)” on page 184.

nvotChangeGraphBackground(3)

Purpose

Changes the background of a graph map

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphBackground (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        char         *         graphBackground)
```

Description

The nvotChangeGraphBackground routine changes the image displayed in the background of the submap into which the graph given by graphProtocol and graphName is exploded.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the graph specified does not exist in the GTM database, its submap does not exist either and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If a graph that matches graphProtocol and graphName but does not have its graphType attribute set to GRAPH exists in the GTM database, its background is not changed and either NVOT_BOX_ALREADY_EXIST or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

Parameters

graphProtocol Specifies the protocol of the root graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the root graph. Both the graphName and graphProtocol parameters are required to uniquely identify the root graph in the GTM database. This parameter is a string of characters used to create the root graph.

graphBackground Specifies an image to be displayed in the background of the submap into which this graph is exploded. Background is usually an image of a geographic region that helps to illustrate a submap. You can select a background image from among the bitmap files in the default directory **/usr/OV/backgrounds**.

Return Values

nvotReturnCode The nvotChangeGraphBackground routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol or name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	A graph does not exist in the GTM database.
[NVOT_BOX_ALREADY_EXIST]	A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.
[NVOT_OTHER_TYPE_GRAPH_EXIST]	Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the background in the submap of the graph created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";

char         *         my_NEW_BackgroundMap  = "usa";


  if ((rc = nvotChangeGraphBackground (mySDLCGraphsProt,
                                       myChildSDLCGraphName,
                                       my_NEW_BackgroundMap)) == NVOT_SUCCESS)

    printf ("Graph background of graph %s changed.\n", myChildSDLCGraphName);
  else
    printf ("Error occurred changing %s background.\n", myChildSDLCGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotCreateBoxInGraph(3)” on page 227.

nvotChangeGraphDetails(3)

Purpose

Changes the contents of the details variable in the database

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphDetails (
                         nvotGraphProtocolType  graphProtocol,
                         char              *    graphName,
                         nvotOctetString   *    graphDetails)
```

Description

The nvotChangeGraphDetails routine changes the contents of the details variable associated with the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the graph specified does not exist in the GTM database, the operation is rejected and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If a graph that matches graphProtocol and graphName but whose graphType attribute is not set to GRAPH exists in the GTM database, its details variable is not changed and either NVOT_BOX_ALREADY_EXIST or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

Parameters

graphProtocol Specifies the protocol of the graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the graph. Both the graphName and graphProtocol parameters are required to uniquely identify the graph in the GTM database. This parameter is a string of characters used to create the graph.

graphDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(graphDetails->octetString, (char *) applStruct, sizeof(applStruct)) and graphDetails->octetLength = sizeof(applStruct). Although nvotOctetString allows for any size strings and the interface does not check the size of graphDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

Return Values

nvotReturnCode The nvotChangeGraphDetails routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol or name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	A graph does not exist in the GTM database.
[NVOT_BOX_ALREADY_EXIST]	A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.
[NVOT_OTHER_TYPE_GRAPH_EXIST]	Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the contents of **myStruct** into the **graphDetails** variable attribute of My_Graph.

```
#include <nvot.h>
typedef struct { int var1, int var2 } structType;

structType            myStruct = { 11, 22 };
nvotOctetString       myGraphDetails;

myGraphDetails.octetString = malloc (sizeof (structType));
myGraphDetails.octetLength = sizeof (structType);

memcpy (myGraphDetails.octetString, (char *) :myStruct, sizeof (structType));

nvotReturnCode         rc;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myGraphName = "My_Graph";

  if ((rc = nvotChangeGraphDetails (myGraphProt,
                                    myGraphName,
                                    :myGraphDetails)) == NVOT_SUCCESS)
```

```
  printf ("myStruct has been stored in %s.\n", myGraphName);
else
  printf ("Error occurred storing myStruct in %s.\n", myGraphName);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.

nvotChangeGraphIcon(3)

Purpose

Changes the icon and label of orphan graphs, boxes, and vertices

Related Functions

nvotChangeGraphLabel
nvotChangeBoxIcon
nvotChangeBoxLabel
nvotChangeVertexIcon
nvotChangeVertexLabel

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphIcon (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        char         *         icon)

nvotReturnCode     nvotChangeGraphLabel (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        char         *         label)

nvotReturnCode nvotChangeBoxIcon (
                        nvotGraphProtocolType  boxProtocol,
                        char *  boxName,
                        char *  icon)

nvotReturnCode nvotChangeBoxLabel (
                        nvotGraphProtocolType boxProtocol,
                        char     *            boxName,
                        char     *            label)

nvotReturnCode     nvotChangeVertexIcon (
                        nvotVertexProtocolType vertexProtocol,
                        char         *         vertexName,
                        char         *         icon)

nvotReturnCode     nvotChangeVertexLabel (
                        nvotVertexProtocolType vertexProtocol,
                        char         *         vertexName,
                        char         *         label)
```

Description

Usually graph, vertex, and box symbol and label information is stored in the member table. This is to allow for the display of a particular symbol on each graph submap where these objects belong.

However, underlying arc endpoint objects might not be members of upper level graphs and therefore do not carry symbol and label information to be displayed in the arc submap. These orphan graphs, boxes,

or vertices carry their symbol and label information in their own primary table (for example, graph and vertex tables).

The protocol and name uniquely identify objects in the GTM database. Thus, *graphProtocol, graphName, boxProtocol, boxName, vertexProtocol, and vertexName* are mandatory parameters. If they are not specified, the operation is not successful and a non-zero return code is set.

If a graph or box does not exist in the GTM database, and the interface has been initialized with CheckOn=TRUE, the icon and label for the object will not be changed, and the error NVOT_GRAPH_DOES_NOT_EXIST or NVOT_BOX_DOES_NOT_EXIST is returned. Automatic creation of graphs is not permitted.

If a vertex does not exist in the GTM database, it is automatically created, and its icon or label is set according to these routines.

Although the icon and label are the target of these routines, they are not mandatory. The icon must be a valid option chosen from the usr/OV/conf/C/oid_to_sym file. If these parameters are not passed, they must be set to NULL. A pointer that is not valid might cause unpredictable errors. If NULL is passed in the icon parameter, the default value **Network:Generic** is used for graphs, **Computer:Generic** is used for boxes, and **Cards:Generic** is used for vertices. If NULL is passed in the label parameter, the graphName, boxName, or vertexName is used.

Parameters

graphProtocol, boxProtocol, vertexProtocol
: Specifies the protocol of the graph, box, or vertex. For more information on specifying a graph protocol, see the /usr/OV/conf/oid_to_protocol file. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

graphName, boxName, vertexName
: Specifies the name of the graph, box, or vertex. It is a string of characters previously used to create the object.

graphIcon, boxIcon, vertexIcon
: Specifies a symbol to represent the graph, box, or vertex on the OVW display. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

graphLabel, boxLabel, vertexLabel
: Specifies the label under the graph, box, or vertex symbol on the OVW display. Label is any string of characters.

Return Values

nvotReturnCode	These routines return an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	The specified graph is not found in the GTM database.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box protocol must be a positive integer and a box name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST]	The specified box is not found in the GTM database.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GTMD_INVALID_RESPONSE]	GTM invalid response. A query to a graph or member table returned an unexpected response from GTMd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateGraph(3)” on page 232.

nvotChangeGraphIconInGraph(3)

Purpose

Changes a graph icon in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphIconInGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  graphProtocol,
                         char         *         graphName,
                         char         *         icon)
```

Description

The nvotChangeGraphIconInGraph routine changes the icon representing the graph identified by graphProtocol and graphName and displayed in the submap of the graph identified by graphProtocolParent and graphNameParent.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, graphProtocol, and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If either the parent or child graph does not exist in the GTM database, the graph icon is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned. Automatic creation of graphs is not supported.

To be supported by the nvotChangeGraphIconInGraph routine, the icon must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if the icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the graph icon is changed to the default symbol **Network:Network**.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in the GTM database. This parameter is a string of characters used to create the parent graph.

graphProtocol Specifies the protocol of the child graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the child graph. Both the graphName and graphProtocol parameters are required to uniquely identify the child graph. This parameter is a string of characters used to create the child graph.

icon Specifies a symbol to represent the child graph in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeGraphIconInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the graph created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";

char         *         mySDLCGraphIcon = "1.3.6.1.2.1.2.2.1.3.10.11";

  if (rc = nvotChangeGraphIconInGraph (mySDLCGraphsProt,
                                       myRootSDLCGraphName,
                                       mySDLCGraphsProt,
                                       myChildSDLCGraphName,
                                       mySDLCGraphIcon) == NVOT_SUCCESS)

    printf ("Graph icon of graph %s changed.\n", myChildSDLCGraphName);
  else
    printf ("An error occurred changing %s icon.\n", myChildSDLCGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotChangeGraphLabelInGraph(3)” on page 181.

nvotChangeGraphLabelInGraph(3)

Purpose

Changes a graph label in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphLabelInGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  graphProtocol,
                         char         *         graphName,
                         char         *         label)
```

Description

The nvotChangeGraphLabelInGraph routine changes the label of the graph identified by graphProtocol and graphName that is displayed in the submap of the graph identified by graphProtocolParent and graphNameParent.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, graphProtocol and graphName parameters are required. If one of these parameters is not provided, NVOT_GRAPH_INVALID_INDEX is returned.

If either the parent or child graph does not exist in GTM database, the graph label is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned. Automatic creation of graphs is not supported.

The label parameter is a character string displayed under a symbol in the NetView for AIX EUI. Usually, it is a human-readable character string that helps to visually identify a resource in a topology map. Although the label must be a valid pointer, NULL is accepted. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the graphName string is displayed in place of the label.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in the GTM database. This parameter is a string of characters used to create the parent graph.

graphProtocol Specifies the protocol of the child graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the child graph. Both the graphName and graphProtocol parameters are required to uniquely identify the child graph. This parameter is a string of characters used to create the child graph.

label Specifies a human-readable character string to be displayed under the graph symbol in the NetView for AIX EUI. It must be a valid character string or NULL.

Return Values

nvotReturnCode The nvotChangeGraphLabelInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the graph created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";

char         *         my_NEW_SDLC_GraphLabel = "My_NEW_SDLC_Graph";

  if (rc = nvotChangeGraphLabelInGraph (mySDLCGraphsProt,
                                        myRootSDLCGraphName,
                                        mySDLCGraphsProt,
                                        myChildSDLCGraphName,
                                        my_NEW_SDLC_GraphLabel) == NVOT_SUCCESS)
```

```
  printf ("Graph icon of graph %s changed.\n", myChildSDLCGraphName);
else
  printf ("An error occurred changing %s icon.\n", myChildSDLCGraphName);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotChangeGraphIconInGraph(3)” on page 178.

nvotChangeGraphPositionInGraph(3)

Purpose

Changes position of a graph icon in a graph submap

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeGraphPositionInGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  graphProtocol,
                         char         *         graphName,
                         nvotPositionType       newPosition)
```

Description

The nvotChangeGraphPositionInGraph routine changes the position of a symbol representing the graph identified by graphProtocol and graphName and associated with the graph identified by graphProtocolParent and graphNameParent.

The parent graph must have been created with layout algorithm set to NONE_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, graphProtocol and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_BOX_INVALID_INDEX is returned.

If the parent graph or the child graph does not exist or they exist but their graphType attribute is not set to GRAPH, the child graph symbol position is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned. Automatic creation of the child graph is not supported.

The nvotPositionType, as defined in the file nvotTypes.h, accepts the following four variables: xCoordinate, yCoordinate, xGrid and yGrid. The xGrid and yGrid variables determine a scale on which the coordinate system is defined.

The grid and coordinate do not necessarily determine the exact physical location in the window where the symbol is displayed. However, they determine a virtual position for the symbol based on the virtual size of the submap.

The symbol size can be affected either by the grid values or by the coordinate values. For example, if the symbol position is set too far from the center or from another symbol, x and y grid are reset to a value that keeps the distances proportional and allows all symbols in the submap to be displayed. This placement of symbols has the effect of a zoomout.

For best results, use the same xGrid and yGrid values for all symbols in the same submap.

Parameters

graphProtocolParent Specifies the protocol of the parent graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph.

graphProtocol Specifies the protocol of the child graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the child graph. Both the graphName and graphProtocol parameters are required to uniquely identify the child graph.

newPosition This parameter is a structure defined in the file nvotTypes.h that specifies the values of the variables xCoordinate, yCoordinate, xGrid and yGrid.

Return Values

nvotReturnCode The nvotChangeGraphPosition routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the position of a child graph symbol.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";

nvotPositionType       myGraphPosition = { 500,  /* xCoordinate */
                                           500,  /* yCoordinate */
                                           1000, /* xGrid       */
                                           1000  /* yGrid       */
                                         };

  if ((rc = nvotChangeGraphPositionInGraph (mySDLCGraphsProt,
                                            myRootSDLCGraphName,
                                            mySDLCGraphsProt,
                                            myChildSDLCGraphName,
                                            myGraphPosition)) == NVOT_SUCCESS)

    printf ("Positioning of graph %s symbol changed.\n", myChildSDLCGraphName);
  else
    printf ("An error occurred changing %s icon position.\n",
                                                          myChildSDLCGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotChangeBoxPositionInGraph(3)” on page 166.

nvotChangeRootGraphIcon(3)

Purpose

Changes a root graph icon

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeRootGraphIcon (
                         nvotGraphProtocolType  graphProtocol,
                         char         *         graphName,
                         char         *         icon)
```

Description

The nvotChangeRootGraphIcon routine changes the icon representing the root graph identified by graphProtocol and graphName and displayed in the NetView for AIX root map.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the root graph specified does not exist in the GTM database, the icon does not exist and the error code NVOT_ROOT_GRAPH_DOES_NOT_EXIST is returned.

If a graph that matches graphProtocol and graphName but is not a root graph exists in the GTM database, its icon is not changed and an error code such as NVOT_GRAPH_ALREADY_EXIST, NVOT_BOX_ALREADY_EXIST, or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

To be supported by the nvotChangeRootGraphIcon routine, the icon parameter must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if the icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the root graph icon is changed to the default symbol **Network:Network**.

Parameters

graphProtocol — Specifies the protocol of the root graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the root graph. Both the graphName and graphProtocol parameters are required to uniquely identify the root graph in the GTM database. This parameter is a string of characters used to create the root graph.

icon — Specifies a symbol to represent the root graph in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode — The nvotChangeRootGraphIcon routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_ROOT_GRAPH_DOES_NOT_EXIST] The root graph does not exist. A root graph must be created before issuing this call.

[NVOT_GRAPH_ALREADY_EXIST] A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_BOX_ALREADY_EXIST] A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_OTHER_TYPE_GRAPH_EXIST] Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the root graph created in the example in “nvotCreateRootGraph(3)” on page 249.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myRootGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myRootGraphName = "My_Root_Graph";
char         *         my_NEW_RootGraphIcon = "1.3.6.1.2.1.2.2.1.3.9.11";

  if ((rc = nvotChangeRootGraphIcon (myRootGraphProt,
                                     myRootGraphName,
                                     my_NEW_RootGraphIcon)) == NVOT_SUCCESS)

    printf ("Graph icon of graph %s changed.\n", myRootGraphName);
  else
```

```
    printf ("An error occurred changing %s icon.\n", myRootGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotChangeRootGraphLabel(3)” on page 190.

nvotChangeRootGraphLabel(3)

Purpose

Changes a root graph label

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeRootGraphLabel (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        char         *         label)
```

Description

The nvotChangeRootGraphLabel routine changes the label under the icon of the root graph identified by graphProtocol and graphName and displayed in the NetView for AIX root map.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the root graph specified does not exist in the GTM database, the label does not exist and the error code NVOT_ROOT_GRAPH_DOES_NOT_EXIST is returned.

If a graph that matches graphProtocol and graphName but is not a root graph exists in the GTM database, its label is not changed and an error code such as NVOT_GRAPH_ALREADY_EXIST, NVOT_BOX_ALREADY_EXIST, or NVOT_OTHER_TYPE_GRAPH_EXIST is returned.

The label parameter is a character string displayed under a symbol in the NetView for AIX EUI. Usually, it is a human-readable character string that helps to visually identify a resource in a topology map. Although the label must be a valid pointer, NULL is accepted. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the graphName string is displayed in place of the label.

Parameters

graphProtocol — Specifies the protocol of the root graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the root graph. Both the graphName and graphProtocol parameters are required to uniquely identify the root graph in the GTM database. This parameter is a string of characters used to create the root graph.

label — Specifies a human-readable character string to be displayed under the root graph symbol in the NetView for AIX EUI. This parameter must be a valid character string or NULL.

Return Values

nvotReturnCode — The nvotChangeRootGraphLabel routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_ROOT_GRAPH_DOES_NOT_EXIST]
The root graph does not exist. A root graph must be created before issuing this call.

[NVOT_GRAPH_ALREADY_EXIST] A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_BOX_ALREADY_EXIST] A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_OTHER_TYPE_GRAPH_EXIST]
Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the root graph created in the example in “nvotCreateRootGraph(3)” on page 249.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myRootGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myRootGraphName = "My_Root_Graph";
char         *         my_NEW_RootGraphLabel = "My_NEW_RootGraphLabel";

  if ((rc = nvotChangeRootGraphLabel (myRootGraphProt,
                                      myRootGraphName,
                                      my_NEW_RootGraphLabel)) == NVOT_SUCCESS)

    printf ("Graph icon of graph %s changed.\n", myRootGraphName);
  else
```

```
    printf ("An error occurred changing %s icon.\n", myRootGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotChangeRootGraphIcon(3)” on page 187.

nvotChangeUnderlyingArcIcon(3)

Purpose

Changes an underlying arc symbol and label

Related Functions

nvotChangeUnderlyingArcLabel

Syntax

```
#include <nvot.h>

nvotReturnCode  nvotChangeUnderlyingArcIcon (
                              nvotNameBindingType    arcNameBindingParent,
                              nvotProtocolType       aEndpointProtocolParent,
                              char *                 aEndpointNameParent,
                              nvotProtocolType       zEndpointProtocolParent,
                              char *                 zEndpointNameParent,
                              int                    arcIndexIdParent,
                              nvotNameBindingType    arcNameBinding,
                              nvotProtocolType       aEndpointProtocol,
                              char *                 aEndpointName,
                              nvotProtocolType       zEndpointProtocol,
                              char *                 zEndpointName,
                              int                    arcIndexId,
                              char *                 ulaIcon)

nvotReturnCode  nvotChangeUnderlyingArcLabel (
                nvotNameBindingType    arcNameBindingParent,
                nvotProtocolType       aEndpointProtocolParent,
                char                  *aEndpointNameParent,
                nvotProtocolType       zEndpointProtocolParent,
                char                  *zEndpointNameParent,
                int                    arcIndexIdParent,
                nvotNameBindingType    arcNameBinding,
                nvotProtocolType       aEndpointProtocol,
                char                  *aEndpointName,
                nvotProtocolType       zEndpointProtocol,
                char                  *zEndpointName,
                int                    arcIndexId,
                char                  *ulaLabel)
```

Description

These routines change the icon and label for an underlying arc. The icon is the symbol used to display the arc on a submap. The label is a human-readable string that is displayed when you press the right mouse button on the symbol of an arc.

The first six parameters identify the parent arc. The next six identify the underlying arc. All these parameters are mandatory.

The parent arc as well as the underlying arc must exist in order for these routines to complete successfully.

Parameters

arcNameBindingParent and *arcNameBinding*
Specifies the class of the objects in each endpoint of the parent arc and the underlying arc, respectively. The endpoint can be a vertex or a graph. The allowed values are:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates that either of the endpoints is a vertex.

ARC_VERTEX_GRAPH_NAME_BINDING
aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either of the endpoints is a graph.

A value other than those in the previous list is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING is set.

a/zEndpointProtocolParent and *a/zEndpointProtocol*
Specifies the protocol of the object identified as the endpoint of the parent arc and the underlying arc, respectively. If *aEndpoint* is to be a vertex, *aEndpointProtocol* must be set to a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, *aEndpoint* is a graph, and *aEndpointProtocol* is a pointer to a valid character string in memory.

a/zEndpointNameParent and *a/zEndpointName*
Specifies the name of the object identified as the endpoint of the parent arc and the underlying arc, respectively. The *endpointName* and *endpointProtocol* are required to identify the object at a certain endpoint of an arc. It must be the same string of characters used in the creation of the underlying arc.

arcIndexIdParent and *arcIndexId*
Specifies indexes (integer values) that distinguish one arc among others between the same endpoints, respectively, of the parent arc and the underlying arc.

It is possible to connect the same two endpoints with several arcs. This parameter provides the means to distinguish between arcs named by the same endpoints.

ulaIcon
Specifies a new symbol to represent this underlying arc in the OVW display. The symbol can be a line, a dotted line, and so on. Refer to the /usr/OV/conf/C/oid_to_sym for details on how to choose an icon.

ulaLabel
Specifies a string of characters that represents the label for the arc. This arc label is displayed in the upper line of the drop down menu that is shown when the right mouse button is clicked on an arc symbol.

Return Values

nvotReturnCode — The nvotChangeUnderlyingArcIcon routine returns an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS] — Successful operation.

[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. An arc protocol must be a positive integer and an arc name must not be NULL.
[NVOT_ULA_INVALID_INDEX]	The ULA index is not valid. A ULA protocol must be a positive integer and a ULA name must not be NULL.
[NVOT_ARC_DOES_NOT_EXIST]	The parent arc for which you are creating an underlying arc does not exist in the GTM database.
[NVOT_INVALID_NAME_BINDING]	Invalid name binding. The name must be a number defined in nvotTypes.h.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

The example below changes one of the underlying arcs created in the example given in the routine “nvotCreateParallelUnderlyingArc(3)” on page 240 with a new symbol and label.

```
#include <nvot.h>

nvotReturnCode      RC;
nvotProtocolType    aEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              aEndpointNameParent = "My_Vertex_V1";
nvotProtocolType    zEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              zEndpointNameParent = "My_Vertex_V2";
int                 arcIndexIdParent = 1;
nvotProtocolType    aEndpointProtocol.vertexProtocol = STARTLAN;
char *              aEndpointName = "My_Vertex_V3";
nvotProtocolType    zEndpointProtocol.vertexProtocol = STARTLAN;
char *              zEndpointName = "My_Vertex_V4";
int                 arcIndexId = 1;
char *              newIcon = "1.3.6.1.2.1.2.2.1.3.54.4";
char *              newLabel = "New_Label_For_V1V2_Ula";

  RC = nvotChangeUnderlyingArcIcon (ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocolParent, aEndpointNameParent,
                                    zEndpointProtocolParent, zEndpointNameParent,
                                    arcIndexIdParent,
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocol, aEndpointName,
                                    zEndpointProtocol, zEndpointName,
                                    arcIndexId, newIcon);

  printf("Change Ula Icon = %s\n", nvotGetErrorMsg(RC));

  RC = nvotChangeUnderlyingArcLabel (ARC_VERTEX_VERTEX_NAME_BINDING,
                                     aEndpointProtocolParent, aEndpointNameParent,
                                     zEndpointProtocolParent, zEndpointNameParent,
```

```
                                    arcIndexIdParent,
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocol, aEndpointName,
                                    zEndpointProtocol, zEndpointName,
                                    arcIndexId, newLabel);

  printf("Change Ula Label = %s\n", nvotGetErrorMsg(RC));
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateSerialUnderlyingArc(3)” on page 253.
- See “nvotCreateParallelUnderlyingArc(3)” on page 240.

nvotChangeVertexDetails(3)

Purpose

Changes the contents of the details variable in the GTM database

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexDetails (
                         nvotVertexProtocolType  vertexProtocol,
                         char              *     vertexName,
                         nvotOctetString   *     vertexDetails)
```

Description

The nvotChangeVertexDetails routine changes the contents of the details variable associated with the vertex identified by vertexProtocol and vertexName.

The protocol and name parameters uniquely identify objects in the GTM database. The vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_VERTEX_INVALID_INDEX is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its details variable set to the value passed in vertexDetails. This is part of GTM's recovery strategy for lost traps.

Parameters

vertexProtocol Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in the GTM database.

vertexDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(vertexDetails->octetString, (char *) applStruct, sizeof(applStruct)) and vertexDetails->octetLength = sizeof(applStruct). Although nvotOctetString allows for any size strings and the interface does not check the size of vertexDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

Return Values

nvotReturnCode The nvotChangeVertexDetails routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the contents of **myStruct** into the **vertexDetails** variable of My_Vertex.

```
#include <nvot.h>
typedef struct { int var1, int var2 } structType;

structType            myStruct = { 11, 22 };
nvotOctetString       myVertexDetails;

myVertexDetails.octetString = malloc (sizeof (structType));
myVertexDetails.octetLength = (sizeof (structType)) };

memcpy (myVertexDetails.octetString, (char *) :myStruct, sizeof (structType));

nvotReturnCode         rc;
nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";

  if ((rc = nvotChangeVertexDetails (myVertexProt,
                                     myVertexName,
                                     :myVertexDetails)) == NVOT_SUCCESS)

    printf ("myStruct has been stored in %s.\n", myVertexName);
  else
    printf ("Error occurred storing myStruct in %s.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.

nvotChangeVertexIconInBox(3)

Purpose

Changes a vertex icon in a box

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexIconInBox (
                         nvotGraphProtocolType  boxProtocol,
                         char         *         boxName,
                         nvotVertexProtocolType vertexProtocol,
                         char         *         vertexName,
                         char         *         icon)
```

Description

The nvotChangeVertexIconInBox routine changes the icon representing the vertex identified by vertexProtocol and vertexName and associated with the box graph identified by boxProtocol and boxName.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol, boxName, vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_BOX_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing box graph does not exist, the vertex icon is not changed and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its icon changed. This is part of GTM's recovery strategy for lost traps.

To be supported by the nvotChangeVertexIconInBox routine, the icon must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if the icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the vertex icon is changed to the default symbol **Cards:Generic**.

Parameters

boxProtocol — Specifies the protocol of the containing box graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName — Specifies the name of the box graph. Both the graphName and graphProtocol parameters are required to uniquely identify the box graph in the GTM database. This parameter is a string of characters used to create the graph.

vertexProtocol — Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in GTM database.

icon Specifies a symbol to represent the vertex in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeVertexIconInBox routine returns an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the vertex created in the example in “nvotCreateVertexInBox(3)” on page 261. Icon is changed to **Cards:Generic**.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBoxName = "My_Box_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexIcon = "1.3.6.1.2.1.2.2.1.3.1.1";

  if (rc = nvotChangeVertexIconInBox (myBoxProt,
                                      myBoxName,
                                      myVertexProt,
                                      myVertexName,
```

```
                                    myVertexIcon) == NVOT_SUCCESS)

  printf ("Vertex icon of vertex %s changed.\n", myVertexName);
else
  printf ("An error occurred changing %s icon.\n", myVertexName);
printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInBox(3)” on page 261.

nvotChangeVertexIconInGraph(3)

Purpose

Changes a vertex icon in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexIconInGraph (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        nvotVertexProtocolType vertexProtocol,
                        char         *         vertexName,
                        char         *         icon)
```

Description

The nvotChangeVertexIconInGraph routine changes the icon representing the vertex identified by vertexProtocol and vertexName and which is associated with the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in GTM database. The graphProtocol, graphName, vertexProtocol and vertexName parameters are required. parameters, If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing graph does not exist, the vertex icon is be changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its icon changed. This is part of GTM's recovery strategy for lost traps.

To be supported by the nvotChangeVertexIconInGraph routine, an icon must be a valid option selected from the file /usr/OV/conf/C/oid_to_sym. However, if an icon is not passed, it must be set to NULL. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the vertex icon is changed to the default symbol **Cards:Generic**.

Parameters

graphProtocol — Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the graph. Both the graphName and graphProtocol parameters are required to uniquely identify the graph in the GTM database. This parameter is a string of characters used to create the graph.

vertexProtocol — Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in GTM database.

icon Specifies a symbol to represent the vertex in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeVertexIconInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the vertex created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         my_STARLAN_GraphName = "My_STARLAN_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexIcon = "1.3.6.1.2.1.2.2.1.3.12.1";

  if (rc = nvotChangeVertexIconInGraph (my_STARLAN_GraphProt,
                                        my_STARLAN_GraphName,
                                        myVertexProt,
                                        myVertexName,
                                        myVertexIcon) == NVOT_SUCCESS)

    printf ("Vertex icon of vertex %s changed.\n", myVertexName);
  else
    printf ("An error occurred changing %s icon.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.

nvotChangeVertexLabelInBox(3)

Purpose

Changes the label of a vertex in a box

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexLabelInBox (
                         nvotGraphProtocolType  boxProtocol,
                         char         *         boxName,
                         nvotVertexProtocolType vertexProtocol,
                         char         *         vertexName,
                         char         *         label)
```

Description

The nvotChangeVertexLabelInBox routine changes the label of a vertex identified by vertexProtocol and vertexName and associated with the box graph identified by boxProtocol and boxName.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol, boxName, vertexProtocol, and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_BOX_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing box graph does not exist, the vertex label is not changed and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its label changed. This is part of GTM's recovery strategy for lost traps.

Label is a character string displayed under a symbol in the NetView for AIX EUI. Usually, it is a human-readable character string that helps to visually identify a network resource. Although the label parameter must be a valid pointer, NULL is accepted. A pointer that is not valid can cause unpredictable errors. If NULL is passed, the vertexName string is displayed in place of the label.

Parameters

boxProtocol — Specifies the protocol of the containing box graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName — Specifies the name of the box graph. Both the graphName and graphProtocol parameters are required to uniquely identify the box graph in the GTM database. This parameter is a string of characters used to create the graph.

vertexProtocol — Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in GTM database.

label — Specifies a symbol to represent the vertex in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeVertexLabelInBox routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the vertex created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBoxName = "My_Box_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexLabel = "My_NEW_STARLAN_Vertex";

  if (rc = nvotChangeVertexLabelInGraph (myBoxProt,
                                         myBoxName,
                                         myVertexProt,
                                         myVertexName,
                                         myVertexLabel) == NVOT_SUCCESS)

    printf ("Vertex label of vertex %s changed.\n", myVertexName);
  else
    printf ("An error occurred changing %s label.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInBox(3)” on page 261.

nvotChangeVertexLabelInGraph(3)

Purpose

Changes the label of a vertex in a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexLabelInGraph (
                        nvotGraphProtocolType  graphProtocol,
                        char         *         graphName,
                        nvotVertexProtocolType vertexProtocol,
                        char         *         vertexName,
                        char         *         label)
```

Description

The nvotChangeVertexLabelInGraph routine changes the label of a vertex identified by vertexProtocol and vertexName and associated with the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in GTM database. The graphProtocol, graphName, vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing graph does not exist, the vertex label is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its label changed. This is part of GTM's recovery strategy for lost traps.

Label is a character string displayed under a symbol in an NetView for AIX window. Usually, it is a human readable character string that helps to visually identify of a network resource. Although the label parameter must be a valid pointer, NULL is accepted. Invalid pointer can cause unpredictable errors. If NULL is passed, the vertexName string is displayed in place of the label.

Parameters

graphProtocol — Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the graph. Both the graphName and graphProtocol parameters are required to uniquely identify the graph in the GTM database. This parameter is a string of characters used to create the graph.

vertexProtocol — Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in GTM database.

label Specifies a symbol to represent the vertex in the NetView for AIX EUI. Valid symbols are defined in the file /usr/OV/conf/C/oid_to_sym.

Return Values

nvotReturnCode The nvotChangeVertexLabelInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the vertex created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         my_STARLAN_GraphName = "My_STARLAN_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexLabel = "My_NEW_STARLAN_Vertex";

  if (rc = nvotChangeVertexLabelInGraph (my_STARLAN_GraphProt,
                                         my_STARLAN_GraphName,
                                         myVertexProt,
                                         myVertexName,
                                         myVertexLabel) == NVOT_SUCCESS)


    printf ("Vertex label of vertex %s changed.\n", myVertexName);
  else
    printf ("An error occurred changing %s label.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.

nvotChangeVertexPositionInBox(3)

Purpose

Changes position of a vertex icon in a box submap

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexPositionInBox (
                         nvotGraphProtocolType  boxProtocol,
                         char         *         boxName,
                         nvotVertexProtocolType vertexProtocol,
                         char         *         vertexName,
                         nvotPositionType       newPosition)
```

Description

The nvotChangeVertexPositionInBox routine changes the position of a symbol representing the vertex identified by vertexProtocol and vertexName and associated with the box graph identified by boxProtocol and boxName.

The containing box must have been created with layout algorithm set to NONE_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol, boxName, vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_BOX_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing box does not exist or it exists but its graphType attribute is not set to BOX, the vertex symbol position is not changed and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values. Its member association to the parent box is also created and its symbol position is set to the values in newPosition. This is part of GTM's recovery strategy for lost traps.

The nvotPositionType, as defined in the file nvotTypes.h, accepts the following four variables: xCoordinate, yCoordinate, xGrid, and yGrid. The variables xGrid and yGrid determine a scale on which the coordinate system is defined.

The grid and coordinate do not necessarily determine the exact physical location in the window where the symbol is displayed. However, they determine a virtual position for the symbol based on the virtual size of the submap.

The symbol size can be affected either by the grid values or by the coordinate values. For example, if the symbol position is set too far from the center or from another symbol, x and y grid are reset to a value that keeps the distances proportional and allows all symbols in the submap to be displayed. This placement of symbols has the effect of a zoomout.

For best results, use the same xGrid and yGrid values for all symbols in the same submap.

Parameters

boxProtocol Specifies the protocol of the containing box. For more information about specifying a box graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the box in the GTM database. This parameter is a string of characters used to create the box.

vertexProtocol Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex. It can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in the GTM database.

newPosition This parameter is a structure defined in the file nvotTypes.h that specifies the values of the variables xCoordinate, yCoordinate, xGrid and yGrid.

Return Values

nvotReturnCode The nvotChangeVertexPositionInBox routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the position of the vertex symbol in the box graph submap.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myBoxGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myBoxGraphName = "My_Box_Graph";

nvotVertexProtocolType myVertexProt = SDLC;
char         *         myVertexName = "My_Vertex";

nvotPositionType       myVertexPosition = { 100,  /* xCoordinate */
                                            100,  /* yCoordinate */
                                            1000, /* xGrid       */
                                            1000  /* yGrid       */
                                          };

  if ((rc = nvotChangeVertexPositionInBox (myBoxGraphProt,
                                           myBoxGraphName,
                                           vertexProt,
                                           vertexName,
                                           myVertexPosition)) == NVOT_SUCCESS)

    printf ("Positioning of vertex %s symbol changed.\n", myVertexName);
  else
    printf ("Error occurred changing %s symbol position.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotChangeVertexPositionInGraph(3)” on page 215.

nvotChangeVertexPositionInGraph(3)

Purpose

Changes position of a vertex icon in a graph submap

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexPositionInGraph (
                         nvotGraphProtocolType  graphProtocol,
                         char         *         graphName,
                         nvotVertexProtocolType vertexProtocol,
                         char         *         vertexName,
                         nvotPositionType       newPosition)
```

Description

The nvotChangeVertexPositionInGraph routine changes the positioning of a symbol representing the vertex identified by vertexProtocol and vertexName and associated with the graph identified by graphProtocol and graphName.

The containing graph must have been created with layout algorithm set to NONE_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol, graphName, vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX is returned.

If the containing graph does not exist or it exists but its graphType attribute is not set to GRAPH, the vertex symbol position is not changed and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values. Its member association to the parent graph is also created and its symbol position is set to the values in newPosition. This is part of GTM's recovery strategy for lost traps.

The nvotPositionType, as defined in the file nvotTypes.h, accepts the following four variables: xCoordinate, yCoordinate, xGrid, and yGrid. The xGrid and yGrid variables determine a scale on which the coordinate system is defined.

The grid and coordinate do not necessarily determine the exact physical location in the window where the symbol is displayed. However, they determine a virtual position for the symbol based on the virtual size of the submap.

The symbol size can be affected either by the grid values or by the coordinate values. For example, if the symbol position is set too far from the center or from another symbol, x and y grid are reset to a value that keeps the distances proportional and allows all symbols in the submap to be displayed. This placement of symbols has the effect of a zoomout.

For best results, use the same xGrid and yGrid values for all symbols in the same submap.

Parameters

graphProtocol Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the graph. Both the graphName and graphProtocol parameters are required to uniquely identify the graph in the GTM database. This parameter is a string of characters used to create the graph.

vertexProtocol Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in GTM database.

newPosition This parameter is a structure defined in the file nvotTypes.h that specifies the values of the variables xCoordinate, yCoordinate, xGrid and yGrid.

Return Values

nvotReturnCode The nvotchangeVertexPositionInGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the position of the vertex in the root graph submap to the upper left corner.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myRootGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myRootGraphName = "My_Root_Graph";

nvotVertexProtocolType myVertexProt = SDLC;
char         *         myVertexName = "My_Vertex";

nvotPositionType       myVertexPosition = { 0,    /* xCoordinate */
                                            0,    /* yCoordinate */
                                            1000, /* xGrid       */
                                            1000  /* yGrid       */
                                          };

  if ((rc = nvotChangeVertexPositionInGraph (myRootGraphProt,
                                             myRootGraphName,
                                             vertexProt,
                                             vertexName,
                                             myVertexPosition)) == NVOT_SUCCESS)

    printf ("Positioning of vertex %s symbol changed.\n", myVertexName);
  else
    printf ("Error occurred changing %s symbol position.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotChangeVertexPositionInBox(3)” on page 212.

nvotChangeVertexStatus(3)

Purpose

Changes the status of a vertex

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotChangeVertexStatus (
                        nvotVertexProtocolType vertexProtocol,
                        char         *         vertexName,
                        statusType             vertexStatus)
```

Description

The nvotChangeVertexStatus routine changes the status of a vertex identified by vertexProtocol and vertexName in the GTM database. This routine consequently changes the color of the symbol representing the vertex in the NetView for AIX EUI.

The protocol and name parameters uniquely identify of objects in the GTM database. The vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the error code NVOT_VERTEX_INVALID_INDEX is returned.

If the vertex does not exist in the GTM database, it is automatically created with default attribute values and has its status changed. This is part of GTM's recovery strategy for lost traps.

The vertexStatus reflects the status of a network resource. The statusType is defined in the file nvotTypes.h. The possible values are mapped into a combination of four status attributes: operational state, alarm status, availability status, and unknown status. For a detailed explanation, see the section about state management variables in the *NetView for AIX Programmer's Guide*. You can handle these status attribute individually through the basic routine calls. If the value passed is not valid, the operation is rejected and error code NVOT_INVALID_STATUS is returned.

Parameters

vertexProtocol — Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in the GTM database.

vertexStatus — Specifies a set of values to represent the status of a resource. This parameter is a combination of MIB variables OperationalState, AlarmStatus, AvailabilityStatus and UnknownStatus. The statusType is defined in the file nvotTypes.h.

Return Values

nvotReturnCode — The nvotChangeVertexStatus routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_INVALID_STATUS]	The status is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example changes the status of the vertex created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
nvotStatusType         myVertexStatus = STATUS_CRITICAL;

  if ((rc = nvotChangeVertexStatus (myVertexProt,
                                    myVertexName,
                                    myVertexStatus) == NVOT_SUCCESS)

    printf ("Vertex status of vertex %s changed.\n", myVertexName);
  else
    printf ("An error occurred changing %s status.\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

/usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.

nvotCreateArcInGraph(3)

Purpose

Creates an arc in a graph

Syntax

```
OVwObjectId     nvotCreateArcInGraph (
                    nvotGraphProtocolType  graphProtocol,
                    char *                 graphName,
                    nvotNameBindingType    arcNameBinding,
                    nvotProtocolType       aEndpointProtocol,
                    char *                 aEndpointName,
                    nvotProtocolType       zEndpointProtocol,
                    char *                 zEndpointName,
                    int                    arcIndexId,
                    char *                 icon,
                    char *                 label,
                    nvotOctetString *      arcDetails,
                    nvotStatusType         status);
```

Description

The nvotCreateArcInGraph routine creates an arc and associate it with a graph. The graph containing the arc must exist. Otherwise the arc will not be created and an error code will be set. An arc can connect two vertices, two graphs, a vertex to a graph, or a graph to a vertex. The vertices and graphs connected by arcs are called *arc endpoints*. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId.

The arcNameBinding parameter helps to identify the arc endpoints. A detailed description follows in the item Parameters. The arcNameBinding must be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of class graph must exist. Otherwise, the arc will not be created and an error code will be set. The GTM interface does not support automatic creation of graphs.

Endpoints of class vertex are automatically created if they do not exist. This is part of the GTM recovery strategy for lost traps. However, a vertex endpoint will NOT be created if the other endpoint refers to a nonexistent graph.

An arc can be a member of several graphs at the same time. If the arc already exists, this routine creates a new association between the arc and a graph. That is, it causes the arc to appear in another graph's submap.

The nvotProtocolType parameter is a union of an enumerated type with a char pointer as defined in <nvotTypes.h> file. Special care must be taken when setting aEndpointProtocol and zEndpointProtocol. Setting these variables to a nvotVertexProtocolType value when arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value for the GTM interface to handle.

The icon, label and arcDetails parameters are the only optional parameters. If they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is passed, the

default **Connection:Generic** symbol is assumed for icon and the concatenation of aEndpointName + zEndpointName + arcIndexId is displayed in place of label.

The status parameter must be set to one of the values defined in the <nvotTypes.h> file. Otherwise, the routine is rejected and the error NVOT_INVALID_STATUS is set. The status value passed to this routine is mapped into other NetView for AIX state values according to the table shown in the *NetView for AIX Programmer's Guide*.

Parameters

graphProtocol
Specifies the protocol of the graph with which this arc is associated. This is the graph of which the arc will be a member. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName
Specifies the name of the graph with which the arc is associated. Both the graphName and the graphProtocol are required to identify the parent graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcNameBinding
Specifies the class of the objects in each endpoint of the arc. The endpoint can be either a vertex or a graph. The supported values are:

ARC_VERTEX_VERTEX_NAME_BINDING
: Indicates that either of the endpoints are vertices

ARC_VERTEX_GRAPH_NAME_BINDING
: Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
: Indicates that aEndpoint is a graph and zEndpoint is a vertex

ARC_GRAPH_GRAPH_NAME_BINDING
: Indicates that either of the endpoint are graphs.

If any value other than those in the preceding list is used, it is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING will be set.

Arcs are handled based on their direction. For more information about the arc direction, see “nvotInit(3)” on page 359. Regardless of the selection made in the initialization session, arcNameBinding always identifies the value set in aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. If the endpoint is a vertex, the endpoint protocol (aEndpointProtocol or zEndpointProtocol) must be set with a value from the enumerated type nvotVertexProtocolType defined in the file <nvotTypes.h>. Otherwise, the endpoint is a graph, and the endpoint protocol must be a pointer to a valid character string in memory.

aEndpointName/zEndpointName
Specifies the name of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. Both the endpoint name and the endpoint protocol are required to identify the object at the aEndpoint or zEndpoint of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) This parameter is an integer value.

icon Specifies the symbol that represents the arc in the NetView for AIX EUI. The symbol can be a line, a dotted line, and so forth. For more information about selecting an icon, see the file /usr/OV/conf/C/oid_to_sym.

label Specifies a string of characters that identifies an arc in the pull-down menu accessed by clicking the right mouse button on an arc symbol.

arcDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(arcDetails->octetString, (char *) applStruct, sizeof(applStruct)) and arcDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of arcDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

status Specifies the status of the arc. Arc status is an enumerated type defined in the file <nvotTypes.h>. For more details, see the *NetView for AIX Programmer's Guide*.

Return Values

OVwObjectId When the application is running in synchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), the nvotCreateArcInGraph routine issues the create arc operation to GTM. The routine remains in a finite loop until the NetView for AIX program returns the OVwObjectId of the arc just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, the routine returns OVwNullObjectId. When the application is running in asynchronous mode (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), the nvotCreateArcInGraph routine issues the create arc operation to GTM and imediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the routine nvotGetError. Refer to “nvotSetSynchronousCreation(3)” on page 368 for more details on OVwObjectId.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol and/or name must not be NULL.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the Z endpoint of the arc does not exist in the GTM database.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

Examples

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

The following example creates one arc between two vertex points. Before creating the arc, you must:

1. Create a root graph (see the example in “nvotCreateRootGraph(3)” on page 249).
2. Create two vertices (V1 and V2) inside the graph (see the example in “nvotCreateVertexInGraph(3)” on page 265).

```
#include <nvot.h>

OVwObjectId            arcId;
nvotReturnCode         rc;
nvotBooleanType        synchMode = FALSE;

/*************** Define the parent graph *********************/
nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Graph";

/*************** Define vertices V1 and V2 *****************/
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;

/*************** Define arc attributes ***************************/
char         *         myLineArcIcon = "1.3.6.1.2.1.2.2.1.3.1.4";
char         *         myLineArcLabel = "My_Line_Arc";
nvotOctetString    *   myLineArcDetails = NULL;
nvotStatusType         myLineArcStatus = STATUS_NORMAL;
int                    arcNumber = 1;

    if (nvotSetSynchronousCreation (TRUE) == NVOT_SUCCESS)
      synchMode = TRUE;

    /* Create one line arc with arcIndexId = 1 */
    if ((arcId = nvotCreateArcInGraph (myGraphProt,
                                       myGraphName,
                                       ARC_VERTEX_VERTEX_NAME_BINDING,
                                       oneEndpoint,
                                       oneEndpointName,
                                       otherEndpoint,
                                       otherEndpointName,
                                       arcNumber,
                                       myLineArcIcon,
                                       myLineArcLabel,
                                       myLineArcDetails,
                                       myLineArcStatus) > OVwNullObjectId)

      printf ("%s OVwObjectId is : %d\n", myLineArcLabel, arcId);
    else
    {
      if (synchMode)
        printf ("An error occurred creating arc %s\n", myLineArcLabel);
    }
    printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotDeleteArcFromGraph(3)” on page 272.
- See “nvotChangeArcIconInGraph(3)” on page 141.
- See “nvotChangeArcLabelInGraph(3)” on page 146.
- See “nvotGetArcsInGraph(3)” on page 307.
- See “nvotSetSynchronousCreation(3)” on page 368.
- See “nvotInit(3)” on page 359.

nvotCreateBoxInGraph(3)

Purpose

Creates a box in a graph

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateBoxInGraph (
                    nvotGraphProtocolType  graphProtocolParent,
                    char *                 graphNameParent,
                    nvotGraphProtocolType  boxProtocol,
                    char *                 boxName,
                    nvotLayoutType         boxLayout,
                    char *                 boxBackground,
                    char *                 icon,
                    char *                 label,
                    nvotOctetString *      boxDetails);
```

Description

The nvotCreateBoxInGraph routine creates a box graph and associates it with a parent graph. A box graph can be a member of more than one parent graph at the same time. Thus, if the box already exists, this routine creates a new association between the box and a parent graph.

The parent, or containing, graph must exist; otherwise, the box graph is not created and an error code is set.

The protocol and name parameters together uniquely identify objects in the gtmd database. These parameters are required for both the parent and box graphs.

The boxLayout parameter is required. However, if -1 (don't care) is passed, NONE_LAYOUT is assumed, any other value is rejected, and the error code NVOT_INVALID_LAYOUT is set. Positioning the symbols in the submap for the vertices and graphs members (submap) of a NONE_LAYOUT box graph requires additional work.

Box background, icon, label and boxDetails are optional parameters. However, if they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is passed, the default **"Computer:Generic"** symbol is assumed for icon and the boxName string is displayed in place of the label.

Parameters

graphProtocolParent — Specifies the protocol of the parent graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent — Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph.

boxProtocol — Specifies the protocol of the child box graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the child box graph. Both the boxName and the boxProtocol parameters are required to uniquely identify the box graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this box graph.

boxLayout Specifies the layout of the child box graph. If -1 is passed, NONE_LAYOUT is assumed. Future changes in the box layout are not supported.

boxBackground Specifies an image to be displayed in the background of the submap into which this box graph is exploded. A background is usually an image of a geographic region that helps to illustrate a submap. You can select a background image from among the bitmap files in the default directory /usr/OV/backgrounds.

icon Specifies the icon to represent this box in the NetView for AIX EUI. For information about selecting an icon, refer to the file /usr/OV/conf/C/oid_to_sym.

label Specifies the label under the box graph icon in the NetView for AIX EUI. The label can be any string of characters.

boxDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(boxDetails->octetString, (char *) applStruct, sizeof(applStruct)) and boxDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of boxDetails, any character exceeding 256 will be truncated by the NetView for AIX object database.

Return Values

OVwObjectId When the application is running in synchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), this routine issues the create box operation to GTM and stands in a finite loop until the NetView for AIX program sends back the OVwObjectId of the box graph just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, this routine returns OVwNullObjectId. When the application is running in asynchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), this routine issues the create box operation to GTM and imediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the nvotGetError routine. For more information about OVwObjectId, see “nvotSetSynchronousCreation(3)” on page 368.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_INVALID_LAYOUT] Invalid layout. The layout must be a number defined in the nvotTypes.h file.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_OVW_OBJECT_ID_NOT_AVAIL]
An error occurred in creating the object ID for this element.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a root graph and then a box graph as member of the root graph.

```
#include <nvot.h>

OVwObjectId            rootGraphId;
OVwObjectId            boxGraphId;
nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";
char         *         myRoot_STARLAN_GraphLabel = "My_Root_STARLAN_Graph";
char         *         myRootGraphBackgroundMap  = "south_america";

char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";
char         *         myBox_STARLAN_GraphLabel = "My_Box_STARLAN_Graph";
char         *         myBoxBackgroundMap  = "brazil";
```

```
nvotLayoutType         my_STARLAN_GraphsLayout = POINT_TO_POINT_RING_LAYOUT;
char         *         my_STARLAN_GraphsIcon = "1.3.6.1.2.1.2.2.1.3.11.10";
nvotOctetString   *    myRootGraphDetails = NULL;
nvotOctetString   *    myChildGraphDetails = NULL;
  rootGraphId = nvotCreateRootGraph (my_STARLAN_GraphsProt,
                                     myRoot_STARLAN_GraphName,
                                     my_STARLAN_GraphsLayout,
                                     myRootGraphBackgroundMap,
                                     my_STARLAN_GraphsIcon,
                                     myRoot_STARLAN_GraphLabel
                                     myRootGraphDetails);
  rc = nvotGetError();
  if ((rc == NVOT_SUCCESS) OR (rc == NVOT_ROOT_GRAPH_ALREADY_EXIST))
  {
    if (synchMode)
      printf ("%s OVwObjectId is : %d\n", myRoot_STARLAN_GraphLabel,
                                                                  rootGraphId);
    else
      printf ("Root graph created but Object Id not available.\n");

    boxGraphId = nvotCreateBoxInGraph (my_STARLAN_GraphsProt,
                                       myRoot_STARLAN_GraphName,
                                       my_STARLAN_GraphsProt,
                                       myBox_STARLAN_GraphName,
                                       my_STARLAN_GraphsLayout,
                                       myBoxBackgroundMap,
                                       my_STARLAN_GraphsIcon,
                                       myBox_STARLAN_GraphLabel
                                       myChildGraphDetails);

    if ((rc = nvotGetError()) == NVOT_SUCCESS)
    {
      if (synchMode)
        printf ("%s OVwObjectId is : %d\n", myBox_STARLAN_GraphLabel,
                                                                    boxGraphId);
      else
        printf ("Box graph created but Object Id not available.\n");
    }
    else
      printf ("An error occurred creating box graph %s\n",
                                                     myBox_STARLAN_GraphLabel);
    printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
  }
  else
    printf ("An error occurred creating root graph %s\n",
                                                    myRoot_STARLAN_GraphLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotDeleteBoxFromGraph(3)” on page 278.
- See “nvotChangeBoxIconInGraph(3)” on page 160.
- See “nvotChangeBoxLabelInGraph(3)” on page 163.
- See “nvotChangeBoxPositionInGraph(3)” on page 166.
- See “nvotGetBoxesInGraph(3)” on page 314.
- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateGraph(3)

Purpose

Creates a graph of graph type GRAPH or BOX

Related Functions

nvotCreateBox

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateGraph (nvotGraphProtocolType  graphProtocol,
                                 char       *           graphName,
                                 nvotLayoutType         graphLayout,
                                 char       *           graphBackground)

OVwObjectId     nvotCreateBox   (nvotGraphProtocolType  boxProtocol,
                                 char       *           boxName,
                                 nvotLayoutType         boxLayout,
                                 char       *           boxBackground)
```

Description

These routines create graphs with graphType attribute set to GRAPH or BOX. They are meant to create graphs and boxes that will become endpoints of underlying arcs. See “nvotCreateSerialUnderlyingArc(3)” on page 253 and “nvotCreateParallelUnderlyingArc(3)” on page 240 for more explanation. Graphs and boxes created through these routines will be much like orphan graphs, in that they will not be correlated to any other parent graph, and will not be displayed before they become underlying arc endpoints unless an explicit call to ***nvotCreateGraphInGraph*** or ***nvotCreateBoxInGraph*** makes reference to these graphs later on.

The protocol and name uniquely identify objects in the GTM database. Both parameters are mandatory.

The *layout* and *background* are also required because they cannot be changed later on.

Parameters

graphProtocol and *boxPrototol*
: Specifies the protocol of the graph or box. For more information on how to specify a graph protocol refer to the /usr/OV/conf/oid_to_protocol file.

graphName and *boxName*
: Specifies the name of the graph or box. The graphName and graphProtocol make up unique information required to identify the graph in the GTM database.

graphLayout and *boxLayout*
: Specifies the layout of the graph or box. If -1 is passed, NONE_LAYOUT is assumed. Note that future changes in the graph layout are not allowed.

graphBackground and *boxBackground*
: Specifies an image to be displayed in the background of the submap into which this graph or box is exploded. Background usually is an image of a geographic region which helps in illustrating a

submap. A background image can be chosen from among the bitmap files in the default directory /usr/OV/backgrounds.

Return Values

OVwObjectId When running the application in synchronous mode, (nvotSetSynchronousCreation with a positive timeout value has been called), this routine will issue the create graph operation to GTM and stand in a finite loop until OVw sends back the OVwObjectId of the graph or box just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, this routine returns ovwNullObjectId. When running the application in asynchronous mode (nvotSetSynchronousCreation with a zero time value has been called), this routine will issue the create graph or create box operation to GTM and immediately return ovwNullObjectId. In either case, upon return, an error code is available through a call to the routine nvotGetError. Refer to “nvotSetSynchronousCreation(3)” on page 368 for more details on OVwObjectId.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box protocol must be a positive integer and a box name must not be NULL.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/oid_to_protocol file.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the example bellow:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example below creates two graphs which are not members of any other graph and which might be endpoints of underlying arcs as shown in the examples given in “nvotCreateSerialUnderlyingArc(3)” on page 253 and “nvotCreateParallelUnderlyingArc(3)” on page 240.

```
#include <nvot.h>

  OVwObjectId            ovwid;

  nvotGraphProtocolType  SDLCGraphProt = "1.3.6.1.2.1.2.2.1.3.17";

  ovwid = nvotCreateGraph (SDLCGraphProt,
                           "SDLC_Endpoint_0",
                           POINT_TO_POINT_LAYOUT,
                           "brazil.gif");

  fprintf(fdOutput, "CreateGraph= %s\n", nvotGetErrorMsg(nvotErrno));
  fprintf(fdOutput, "SDLC_Endpoint_0 = %d\n\n",ovwid);

  ovwid = nvotCreateGraph (SDLCGraphProt,
                           "SDLC_Endpoint_1",
                           POINT_TO_POINT_LAYOUT,
                           "argentina.gif");

  fprintf(fdOutput, "CreateGraph= %s\n", nvotGetErrorMsg(nvotErrno));
  fprintf(fdOutput, "SDLC_Endpoint_1 = %d\n\n",ovwid);
```

Libraries

- /usr/OV/lib/libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateParallelUnderlyingArc(3)” on page 240
- See “nvotCreateSerialUnderlyingArc(3)” on page 253

nvotCreateGraphInGraph(3)

Purpose

Creates a graph in a graph

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateGraphInGraph (
                    nvotGraphProtocolType  graphProtocolParent,
                    char *                 graphNameParent,
                    nvotGraphProtocolType  graphProtocol,
                    char *                 graphName,
                    nvotLayoutType         graphChildLayout,
                    char *                 graphChildBackground,
                    char *                 icon,
                    char *                 label,
                    nvotOctetString *      graphChildDetails);
```

Description

The nvotCreateGraphInGraph routine creates a graph and associates it with a parent graph. A child graph can be a member of several parent graphs at the same time. If the child graph already exists, the nvotCreateGraphInGraph routine creates a new association between the child graph and a parent graph so that the child graph is displayed on another parent graph's submap.

The parent, or containing, graph must exist; otherwise, the child graph is not created and an error code is set. The protocol and name parameters together uniquely identify objects in the GTM database. These parameters are required for both parent and child graphs.

The graphChildLayout parameter is required. If -1 (don't care) is passed, NONE_LAYOUT is assumed. any other value is rejected, and the error code NVOT_INVALID_LAYOUT is set. Positioning the symbols in the submap for the vertices and graphs members (submap) of a NONE_LAYOUT graph requires additional work. Also, further changes to the graph layout attribute are not supported.

The graphChildBackground, icon, label and graphChildDetails parameters are the only optional parameters. If they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is passed, the default “Network:Network” symbol is assumed for icon and the graphName string is displayed in place of the label.

Parameters

graphProtocolParent Specifies the protocol of the parent graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are to uniquely identify the parent graph.

graphProtocol Specifies the protocol of the child graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the child graph. Both the graphName and graphProtocol parameters are required to uniquely identify the child graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

graphChildLayout Specifies the layout of the child graph. If -1 is passed, NONE_LAYOUT is assumed. Future changes in the graph layout are not supported.

graphChildBackground Specifies an image to be displayed in the background of the submap into which this child graph is exploded. A background is usually an image of a geographic region that helps to illustrate a submap. You can select a background image from among the bitmap files in the default directory /usr/OV/backgrounds.

icon Specifies the icon to represent the child graph in the NetView for AIX EUI. For more information about selecting an icon, refer to the file /usr/OV/conf/C/oid_to_sym.

label Specifies the label under the graph icon in the NetView for AIX EUI. A label can be any string of characters.

graphChildDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(graphChildDetails->octetString, (char *) applStruct, sizeof(applStruct)) and graphChildDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of graphChildDetails, any character exceeding 256 will be truncated by the NetView for AIX object database.

Return Values

OVwObjectId When the application is running in synchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), the nvotCreateGraphInGraph routine issues the create graph operation to GTM and stands in a finite loop until the NetView for AIX program returns the OVwObjectId of the graph just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, this routine returns OVwNullObjectId. When the application is running in asynchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), the nvotCreateGraphInGraph routine issues the create graph operation to GTM and immediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the nvotGetError routine. For more information about OVwObjectId, see “nvotSetSynchronousCreation(3)” on page 368.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_INVALID_LAYOUT] Invalid layout. The layout must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_OVW_OBJECT_ID_NOT_AVAIL]
An error occurred in creating the object ID for this element.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a root graph and then a child graph as member of the root graph.

```
#include <nvot.h>

OVwObjectId            rootGraphId;
OVwObjectId            childGraphId;
nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myRootSDLCGraphLabel = "My_Root_SDLC_Graph";
char         *         myRootGraphBackgroundMap  = "south_america";

char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";
char         *         myChildSDLCGraphLabel = "My_Child_SDLC_Graph";
char         *         myChildGraphBackgroundMap  = "brazil";

nvotLayoutType         mySDLCGraphsLayout = POINT_TO_POINT_RING_LAYOUT;
char         *         mySDLCGraphsIcon = "1.3.6.1.2.1.2.2.1.3.11.11";
nvotOctetString   *    myRootGraphDetails = NULL;
nvotOctetString   *    myChildGraphDetails = NULL;
```

```
rootGraphId = nvotCreateRootGraph (mySDLCGraphsProt,
                                   myRootSDLCGraphName,
                                   mySDLCGraphsLayout,
                                   myRootGraphBackgroundMap,
                                   mySDLCGraphsIcon,
                                   myRootSDLCGraphLabel
                                   myRootGraphDetails);
rc = nvotGetError();
if ((rc == NVOT_SUCCESS) OR (rc == NVOT_ROOT_GRAPH_ALREADY_EXIST))
{
  if (synchMode)
    printf ("%s OVwObjectId is : %d\n", myRootSDLCGraphLabel, rootGraphId);
  else
    printf ("Root graph created but Object Id not available.\n");

  childGraphId = nvotCreateGraphInGraph (mySDLCGraphsProt,
                                         myRootSDLCGraphName,
                                         mySDLCGraphsProt,
                                         myChildSDLCGraphName,
                                         mySDLCGraphsLayout,
                                         myChildGraphBackgroundMap,
                                         mySDLCGraphsIcon,
                                         myChildSDLCGraphLabel
                                         myChildGraphDetails);
  if ((rc = nvotGetError()) == NVOT_SUCCESS)
  {
    if (synchMode)
      printf ("%s OVwObjectId is : %d\n", myChildSDLCGraphLabel, childGraphId);
    else
      printf ("Child graph created but Object Id not available.\n");
  }
  else
    printf ("An error occurred creating graph %s\n", myChildSDLCGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
}
else
  printf ("An error occurred creating root graph %s\n", myRootSDLCGraphLabel);
printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotDeleteGraphFromGraph(3)” on page 283.
- See “nvotChangeGraphIconInGraph(3)” on page 178.
- See “nvotChangeGraphLabelInGraph(3)” on page 181.
- See “nvotChangeGraphPositionInGraph(3)” on page 184.
- See “nvotGetGraphsInGraph(3)” on page 330.

- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateParallelUnderlyingArc(3)

Purpose

Creates an arc that lies under another arc

Syntax

```
nvotReturnCode  nvotCreateParallelUnderlyingArc (
                    nvotNameBindingType    arcNameBindingParent,
                    nvotProtocolType       aEndpointProtocolParent,
                    char        *          aEndpointNameParent,
                    nvotProtocolType       zEndpointProtocolParent,
                    char        *          zEndpointNameParent,
                    int                    arcIndexIdParent,
                    nvotNameBindingType    arcNameBinding,
                    nvotProtocolType       aEndpointProtocol,
                    char        *          aEndpointName,
                    nvotProtocolType       zEndpointProtocol,
                    char        *          zEndpointName,
                    int                    arcIndexId,
                    char        *          ulaIcon,
                    char        *          ulaLabel)
```

Description

An arc symbol in the ovw screen can actually be made up of several underlying arcs. NetView for AIX open topology provides the *UnderlyingArc* table as a means for storing underlying arcs in the database and displaying them in the ovw display.

Underlying arcs can be connected to each other either in a serial or parallel fashion. This routine creates an arc that lies under a parent arc and is displayed in parallel to other underlying arcs at the same level.

The first six parameters identify the parent arc. The parent arc must exist before the underlying arc can be created. If the parent arc does not exist, NVOT_ARC_DOES_NOT_EXIST is returned.

The arcNameBinding, aEndpointProtocol, aEndpointName, zEndpointProtocol, zEndpointName and arcIndexId parameters identify the underlying arc.

Note: There is a possibility that the underlying arc will have the same name as its parent arc, such as when their endpoints are actually the same and their indexIds have erroneously been set to the same value. This error should be avoided because the NVOT API does not check for this, but gtmd does and it will reject the trap.

The arcNameBinding parameter helps in identifying the uderlying arc endpoints. The arcNameBinding value must be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of the class graph must exist. If they do not, the underlying arc is not created and an error code is set. The NVOT API does not allow for automatic creation of graphs or boxes.

Usually, graphs and boxes are created as members of upper-level graphs through calls to routines *nvotCreateGraphInGraph* and *nvotCreateBoxInGraph*. However, most of the time, the graph or box endpoints of underlying arcs are not members of any upper-level graph. In other words, they exist as

orphan graphs or boxes and are displayed only in the arc submap. Because the automatic creation of graphs or boxes is not allowed, two routines are available to allow for the creation of these orphan graphs or boxes (refer to “nvotCreateGraph(3)” on page 232).

Endpoints of class vertex are automatically created if they do not exist. This is part of the GTM recovery strategy for lost traps. However, a vertex endpoint is *not* created if the other endpoint is a reference to a nonexistent graph.

The ulaIcon and ulaLabel parameters are optional. If they are not passed, they must be set to NULL. Pointers that are not valid can cause unpredictable errors. If the parameters are set to NULL, the default **Connection:Generic** symbol is assumed for ulaIcon, and the concatenation of aEndpointName + zEndpointName + arcIndexId is displayed in place of ulaLabel.

Parameters

arcNameBindingParent and *arcNameBinding*
Specifies the class of the objects in each endpoint of the parent arc and the underlying arc, respectively. The endpoint can be a vertex or a graph. The allowed values are as follows:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates either of the endpoints are vertices.

ARC_VERTEX_GRAPH_NAME_BINDING
aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either of the endpoints are graphs.

Any value other than the above is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING is set.

a/zEndpointProtocolParent and *a/zEndpointProtocol*
Specifies the protocol of the object identified as the endpoint, respectively, of the parent arc and the underlying arc. If aEndpoint is to be a vertex, aEndpointProtocol must be set to a value from the enumerated type nvotVertexProtocolType defined in the the file <nvotTypes.h>. Otherwise, aEndpoint is a graph, and aEndpointProtocol is a pointer to a valid character string in memory.

a/zEndpointNameParent and *a/zEndpointName*
Specifies the name of the object identified as the endpoint of the parent arc and the underlying arc, respectively. The endpointName and endpointProtocol are required to identify the object at a certain endpoint of an arc. The name can be any string of characters. However, once specified, the same name must be used in any reference to the object.

IndexIdParent and *arcIndexId*
Specifies indexes (integer values) that distinguish an arc from others between the same endpoints of the parent arc and the underlying arc, respectively.

It is possible to connect the same two endpoints with several arcs; this variable is provided to give you a means for distinguishing between arcs named by the same endpoints.

ulaIcon
Specifies the symbol to represent this underlying arc in OVw display. The symbol may be a line, a dotted line, and so on. Refer to the /usr/OV/conf/C/oid_to_sym file for details on how to choose an icon.

ulaLabel
Specifies a string of characters that represent an arc label to be displayed in the drop down menu shown when the right mouse button is clicked on an arc symbol. This is also true for an underlying arc.

Return Values

nvotReturnCode — The nvotCreatParallelUnderlyingArc routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. An arc protocol must be a positive integer and an arc name must not be NULL.
[NVOT_ULA_INVALID_INDEX]	The ULA index is not valid. A ULA protocol must be a positive integer and a ULA name must not be NULL.
[NVOT_ENDPOINT_INVALID_INDEX]	The endpoint index is not valid. An endpoint protocol must be a positive integer and an endpoint name must not be NULL.
[NVOT_ARC_DOES_NOT_EXIST]	The parent arc for which you are creating an underlying arc does not exist in the GTM database.
[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The aEndPoint graph specified for the underlying arc does not exist in the GTM database.
[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The zEndPoint graph specified for the underlying arc does not exist in the GTM database.
[NVOT_INVALID_NAME_BINDING]	The name binding is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
This example illustrates how to create two arcs in parallel underlying
an arc previously created.

#include <nvot.h>

nvotReturnCode      RC;
nvotProtocolType    aEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              aEndpointNameParent = "My_Vertex_V1";
nvotProtocolType    zEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              zEndpointNameParent = "My_Vertex_V2";
int                 arcIndexIdParent = 1;
nvotProtocolType    aEndpointProtocol1.vertexProtocol = STARTLAN;
char *              aEndpointName1 = "My_Vertex_V3";
nvotProtocolType    zEndpointProtocol1.vertexProtocol = STARTLAN;
char *              zEndpointName1 = "My_Vertex_V4";
int                 arcIndexId1 = 1;
nvotProtocolType    aEndpointProtocol2.vertexProtocol = STARTLAN;
char *              aEndpointName2 = "My_Vertex_V5";
nvotProtocolType    zEndpointProtocol2.vertexProtocol = STARTLAN;
char *              zEndpointName2 = "My_Vertex_V6";
int                 arcIndexId1 = 1;
char *              icon = "1.3.6.1.2.1.2.2.1.3.54.4";

  RC = nvotCreateParallelUnderlyingArcIcon (
                                  ARC_VERTEX_VERTEX_NAME_BINDING,
                                  aEndpointProtocolParent, aEndpointNameParent,
                                  zEndpointProtocolParent, zEndpointNameParent,
                                  arcIndexIdParent,
                                  ARC_VERTEX_VERTEX_NAME_BINDING,
                                  aEndpointProtocol1, aEndpointName1,
                                  zEndpointProtocol1, zEndpointName1,
                                  arcIndexId1, icon, "Ula_V1V2_1");

  printf("Create First Parallel Ula = %s\n", nvotGetErrorMsg(RC));


  RC = nvotCreateParallelUnderlyingArcIcon (
                                  ARC_VERTEX_VERTEX_NAME_BINDING,
                                  aEndpointProtocolParent, aEndpointNameParent,
                                  zEndpointProtocolParent, zEndpointNameParent,
                                  arcIndexIdParent,
                                  ARC_VERTEX_VERTEX_NAME_BINDING,
                                  aEndpointProtocol2, aEndpointName2,
                                  zEndpointProtocol2, zEndpointName2,
                                  arcIndexId2, icon, "Ula_V1V2_2");

  printf("Create Second Parallel Ula = %s\n", nvotGetErrorMsg(RC));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.
- See “nvotCreateSerialUnderlyingArc(3)” on page 253.

nvotCreateProvidingSap(3)

Purpose

Creates a SAP of SAP type PROVIDING

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotCreateProvidingSap (
                       nvotVertexProtocolType  vertexProtocol,
                       char        *           vertexName,
                       nvotVertexProtocolType  sapProtocol,
                       char        *           sapName);
```

Description

Vertices represent communication entities or interfaces across various protocol layers. The SAP object class represents the logical relationship between two vertices inside a computer. If a communication entity in a given protocol layer uses the services of a lower layer entity through a service point, a vertex representing an *N*-layer entity uses a SAP provided by a vertex representing an entity in layer *N-1*. Likewise, a vertex representing an entity in layer *N* can provide a SAP for use by other vertices representing entities in layer *N+1*. An interface or communication entity can provide its services to more than one entity in an upper layer at the same time. However, a SAP always establishes only one association.

In terms of open topology map representation, the SAP object creation is a means to correlate vertex symbols in box submaps.

The following rules apply to correlation of vertex symbols and representation of submaps and symbols:

- Although a vertex can be a member of a graph of type GRAPH, vertices correlated by a SAP should be entities running inside the same computer. This computer is represented by a graph of type BOX. It is recommended that you use SAPs to correlated vertices that are members of BOXES. Although the interface does not check whether the vertex referenced by the SAP is a member of a GRAPH, using a vertex that is a member of a graph of type GRAPH might produce unpredictable results.
- To avoid a rapid increase in the database, open topology merges vertices' objects' information into one object when a SAP is created. Some of the information merged is based on the protocol used. The SAP creation must take different protocol values in the variables *vertexProtocol* and *sapProtocol*. Otherwise, the vertex symbol referenced by *sapProtocol* and *sapName* disappears from the display.

Saps can be used to correlate either non-IP/IP or non-IP/non-IP objects:

- NonIP/IP correlation:

  IP topology might have discovered a node running IP on top of a given interface card. Consider that a non-IP management application is to represent an entity of its own protocol such as an interface card, providing its services to an IP entity. But, the non-IP application does not yet know of the existence of the IP side. The non-IP application is to provide correlation. Given that the non-IP management application has already created a vertex V1 to represent its entity card, a SAP to correlate non-IP vertex V1 with the interface card already discovered by the IP side would look like this:

  vertexProtocol Set to the value of V1 protocol.

  vertexName Set to the value of V1 name.

sapProtocol Set the value of the protocol defined in the interface card or, if the protocol is not known, use OTHER_PROTOCOL as defined in the file nvotTypes nvotVertexProtocolType. To avoid breaking the preceding rule (see the rule on page 245), the field must not assume the same value of the *vertexProtocol*.

sapName Set to the value of the universal address of the interface card. For the correlation to take effect for a given computer, this field must be set to the value set in the field `SNMP ifPhysAddr` of the corresponding IP node. If the interface is a Token Ring card, for example, this field is set to its MAC address.

- Non-IP/non-IP correlation:

  Different protocol BOX graphs B1 and B2 contain, respectively, vertices V1 and V2. A SAP correlation of these vertices V1 and V2 indicates that they are running in the same box. This means that boxes B1 and B2 would be the same computer. So, open topology would merge the information of B1 and B2. This case is similar to the non-IP/IP correlation except that, instead of a non-IP box graph and an IP node being correlated, two non-IP box graphs are correlated.

  Another case is a single BOX graph that contains two vertices V1 and V2. In both cases, a SAP to correlated vertices V1 and V2 would be:

  vertexProtocol Set to the value of V1 protocol.

  vertexName Set to the value of V1 name.

  sapProtocol Set to the value of V2 protocol.

  sapName Set to the value of V2 name.

  For example, a Token Ring interface can provide its services to SNA Services and TCP/IP stacks inside the same box. The LLC layer entity would provide two distinct SAPs, one for SNA Services and another for TCP/IP. The example in this man page illustrates such a piece of topology.

  The nvotCreateProvidingSap routine creates a SAP uniquely identified by the values of the sapProtocol and sapName parameters. The vertex identified by vertexProtocol and vertexName uses the service of this SAP.

  If the vertex providing this SAP does not yet exist in the gtmd database, it is automatically created. However, the automatic creation of a vertex requires future calls to routines nvotChangeVertexIconInBox, nvotChangeVertexLabelInBox, nvotChangeVertexIconInGraph or nvotChangeVertexLabelInGraph for accurate display by the NetView for AIX program.

  It does not make sense to create a using SAP when a providing SAP does not exist. If you create a using SAP when a providing SAP does not exist, and later issue a request to create a providing SAP with reference to the SAP created (using the same sapProtocol and sapName values), a new SAP will not be created. Instead, the request will cause two vertices to be associated through a common SAP.

All parameters in this routine are required.

Parameters

vertexProtocol Specifies the protocol of the vertex providing the SAP. The vertex protocol is an enumerated type defined in the file <nvotTypes.h>.

vertexName Specifies the name of the vertex providing the SAP. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this vertex.

sapProtocol Specifies the protocol of the vertex in which the SAP is defined. This is the protocol of the vertex providing this SAP. For more information, see the example in this man page.

sapName The sapName or sapAddressName parameter identifies a SAP provided by an *N*-level entity to an *N+1*-level entity. This parameter is a character string containing an IP address, a SNA physical and logical unit address, and so on. For more information, see the example in this man page.

Return Values

nvotReturnCode The nvotCreateProvidingSap routine returns an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example illustrates how to represent a SAP provided by a Token Ring interface card.

```
#include <nvot.h>

nvotReturnCode         rc;

/************ Define LAN vertex (V1) **********************/
nvotVertexProtocolType myLNM_Prot = LANBRIDGE;
char         *         myLNM_Name = "LAN_Vertex";

/************ Define Token Ring SAP ***********************/
nvotVertexProtocolType myTokenRingProt = ISO88025_TOKENRING;
char         *         myTokenRingAddr = "10005AA8D718";

  if ((rc = nvotCreateProvidingSap (myLNM_Prot,
                                    myLNM_Name,
                                    myTokenRingProt,
```

```
                                    myTokenRingAddr)) == NVOT_SUCCESS)

   printf ("Sap created successfully");
 else
   printf ("Error : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateRootGraph(3)

Purpose

Creates a root graph

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateRootGraph (
                    nvotGraphProtocolType  graphProtocol,
                    char *                 graphName,
                    nvotLayoutType         graphLayout,
                    char *                 graphBackground,
                    char *                 icon,
                    char *                 label,
                    nvotOctetString *      graphDetails);
```

Description

The nvotCreateRootGraph creates a root graph. A root graph is not a member of any other graph. This routine does not support the changing of a graph or box into a root graph.

In the following cases, a root graph is not created, an error code is returned, and the interface returns OVwNullObjectId.

- A non-root graph or box that matches graphProtocol and graphName exists in the GTM database. In this case, the error code NVOT_GRAPH_ALREADY_EXIST or NVOT_BOX_ALREADY_EXIST is set.
- A root graph that matches graphProtocol and graphName exists. In this case, the error code NVOT_ROOT_GRAPH_ALREADY_EXIST is set.
- A graph with the graphType attribute set to INVALID_GRAPH or OTHER_GRAPH already exists in GTM database. In this case, the error code NVOT_OTHER_TYPE_GRAPH_EXISTS is set.

The graphProtocol and graphName parameters are required because together they uniquely identify graphs in the GTM database.

The graphLayout parameter is required. However, if -1 (don't care) is passed, NONE_LAYOUT is assumed, any other value is rejected, and the error code NVOT_INVALID_LAYOUT is set. Positioning the symbols in the submap for the vertices and graphs members (submap) of a NONE_LAYOUT root graph requires additional work. Also, further changes to the graph layout attribute are not supported.

The graphBackground, icon, label and graphDetails are the only optional parameters. However, if they are not passed, they must be set to NULL. Pointers that are not valid can cause unpredictable errors. If NULL is passed, the default “Network:Network” symbol is assumed for icon and the graphName string is displayed in place of the label. Also, the submap background is cleared.

Parameters

graphProtocol Specifies the protocol of the root graph. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the root graph. Both the graphName and the graphProtocol are required to uniquely identify the root graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

graphLayout Specifies the layout of the submap into which this graph can be exploded in the NetView for AIX EUI. This parameter is an enumerated type defined in the file <nvotTypes.h>. NONE_LAYOUT is assumed if -1 is passed. Once the graph is created, the graph layout attribute cannot be changed.

graphBackground Specifies an image to be displayed in the background of the submap into which this root graph is exploded. A background is usually an image of a geographic region that helps in illustrating a submap. You can select a background image from among the bitmap files in the default directory /usr/OV/backgrounds.

icon Specifies the icon to represent this root graph in the NetView for AIX EUI. For information about selecting an icon, refer to the file /usr/OV/conf/C/oid_to_sym.

label Specifies the label under the root graph icon in the NetView for AIX EUI. The label can be any string of characters.

graphDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(graphDetails->octetString, (char *) applStruct, sizeof(applStruct)) and graphDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of graphDetails, any character exceeding 256 will be truncated by the NetView for AIX object database.

Return Values

OVwObjectId When the application is running in synchronous mode (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), the nvotCreateRootGraph routine issues the create root graph operation to GTM and stands in a finite loop until the NetView for AIX program returns the OVwObjectId of the root graph just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, the routine returns OVwNullObjectId. When the application is running in asynchronous mode (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), the nvotCreateRootGraph routine issues the create root graph operation to GTM and immediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the routine nvotGetError. For more information about OVwObjectId, see “nvotSetSynchronousCreation(3)” on page 368.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_INVALID_LAYOUT] Invalid layout. The layout must be a number defined in the nvotTypes.h file.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_ALREADY_EXIST] A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_BOX_ALREADY_EXIST] A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_OTHER_TYPE_GRAPH_EXIST]
Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.

[NVOT_ROOT_GRAPH_ALREADY_EXIST]
A root graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_OVW_OBJECT_ID_NOT_AVAIL]
An error occurred in creating the object ID for this element.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a graph as root of protocol SDLC and layout POINT_TO_POINT_RING_LAYOUT.

```
#include <nvot.h>

OVwObjectId            rootGraphId;
nvotReturnCode         rc;

nvotGraphProtocolType  myRootGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myRootGraphName = "My_Root_Graph";
nvotLayoutType         myRootGraphLayout = POINT_TO_POINT_RING_LAYOUT;
char         *         myRootGraphBackgroundMap = "brazil";
char         *         myRootGraphIcon = "1.3.6.1.2.1.2.2.1.3.11.11";
char         *         myRootGraphLabel = "My_Root_SDLC_Graph";
nvotOctetString   *    myRootGraphDetails = NULL;
  rootGraphId = nvotCreateRootGraph (myRootGraphProt,
                                     myRootGraphName,
                                     myRootGraphLayout,
                                     myRootGraphBackgroundMap,
                                     myRootGraphIcon,
                                     myRootGraphLabel,
                                     myRootGraphDetails);
  if ((rc = nvotGetError()) == NVOT_SUCCESS)
  {
    if (synchMode)
      printf ("%s OVwObjectId is : %d\n", myRootGraphLabel, rootGraphId);
    else
      printf ("Root graph created but Object Id not available.\n");
  }
  else
  {
    printf ("Root graph may not have been created.\n");
  }
  printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotChangeRootGraphLabel(3)” on page 190.
- See “nvotChangeRootGraphIcon(3)” on page 187.
- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateSerialUnderlyingArc(3)

Purpose

Creates an arc that lies under another arc

Syntax

```
nvotReturnCode  nvotCreateSerialUnderlyingArc (
                    nvotNameBindingType    arcNameBindingParent,
                    nvotProtocolType       aEndpointProtocolParent,
                    char        *          aEndpointNameParent,
                    nvotProtocolType       zEndpointProtocolParent,
                    char        *          zEndpointNameParent,
                    int                    arcIndexIdParent,
                    nvotNameBindingType    arcNameBinding,
                    nvotProtocolType       aEndpointProtocol,
                    char        *          aEndpointName,
                    nvotProtocolType       zEndpointProtocol,
                    char        *          zEndpointName,
                    int                    arcIndexId,
                    char        *          ulaIcon,
                    char        *          ulaLabel,
                    nvotNameBindingType    nextSerialNameBinding,
                    nvotProtocolType       nextSerialAEndpointProtocol,
                    char        *          nextSerialAEndpointName,
                    nvotProtocolType       nextSerialZEndpointProtocol,
                    char        *          nextSerialZEndpointName,
                    int                    nextSerialArcIndexId)
```

Description

An arc symbol in the OVw screen can actually be a bundle of arcs. NetView for AIX open topology provides the *UnderlyingArc* table as a means for storing lower level arcs in the database and representing them on the OVw display.

Underlying arcs can be connected to each other either in a serial or parallel fashion. This routine creates an arc that lies under a parent arc and is to be displayed as serially connected.

The first six parameters identify the parent arc. The parent arc must exist before the underlying arc can be created. If the parent arc does not exist, NVOT_ARC_DOES_NOT_EXIST is returned.

The arcNameBinding, aEndpointProtocol, aEndpointName, zEndpointProtocol, zEndpointName and arcIndexId parameters describe the underlying arc itself.

Note: There is a possibility that the underlying arc will have the same name as its parent arc, such as when their endpoints are actually the same and their indexIds have erroneously been set to the same value. This error should be avoided because the NVOT API does not check for this, but gtmd does and it will reject the trap.

The arcNameBinding parameter helps in identifying the underlying arc endpoints. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

Endpoints of the class graph must exist; otherwise, the underlying arc is not created and an error code is set. The NVOT API does not allow for automatic creation of graphs.

Usually, graphs and boxes are created as members of upper-level graphs through calls to routines *nvotCreateGraphInGraph* and *nvotCreateBoxInGraph*. However, most of the time, the graph or box endpoints of underlying arcs are not members of any upper-level graph. In other words, they exist as orphan graphs or boxes and are displayed only in the arc submap. Because the automatic creation of graphs or boxes is not allowed, two routines are available to allow for the creation of these orphan graphs or boxes (refer to “nvotCreateGraph(3)” on page 232).

Endpoints of class vertex are automatically created if they do not exist. This is part of the GTM recovery strategy for lost traps. However, a vertex endpoint is *not* created if the other endpoint is a reference to a nonexistent graph.

The ulaIcon and ulaLabel parameters are optional. If they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is set for the parameters, the default **Connection:Generic** symbol is assumed for ulaIcon, and the concatenation of aEndpointName + zEndpointName + arcIndexId is displayed in place of ulaLabel.

The last six parameters specify the nextSerial underlying arc. The symbol for this arc is displayed next to the underlying arc being created. If this is the last underlying arc in a series, the nextSerial parameters must not be filled in and *nextSerialNameBinding* must be set to DONT_CARE_NAME_BINDING. If the nextSerial parameter makes reference to a non-existent arc, this underlying arc is not created and NVOT_NEXT_SERIAL_ARC_DOES_NOT_EXIST is returned. The next serial arc is not automatically created by gtmd.

Parameters

arcNameBindingParent, *arcNameBinding*, and *nextSerialNameBinding*
Specifies the class of the objects in each endpoint, respectively, of the parent arc, the underlying arc, and of the next arc in the series. The endpoint can be a vertex or a graph. The allowed values are:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates one of the endpoints is a vertex.

ARC_VERTEX_GRAPH_NAME_BINDING
Indicates aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
Indicates aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates one of the endpoints is a graph.

In addition to the previously listed four values, the nextSerialNameBinding also supports the DONT_CARE_NAME_BINDING value. Any value other than those noted is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING is set.

a/zEndpointProtocolParent, *a/zEndpointProtocol*, and *nextSerialA/ZEndpointProtocol*
Specifies the protocol of the object identified as the endpoint of the parent arc, the underlying arc, and next arc in the series, respectively. If aEndpoint is a vertex, aEndpointProtocol must be set to a value from the enumerated type nvotVertexProtocolType defined in the the file <nvotTypes.h>. Otherwise, aEndpoint is a graph, and aEndpointProtocol is a pointer to a valid character string in memory.

a/zEndpointNameParent, *a/zEndpointName*, and *nextSerialA/ZEndpointName*
Specifies the name of the object identified as the endpoint of the parent arc, the underlying arc, and the next serial arc, respectively. The endpointName, together with endpointProtocol, provides the

information required to identify the object at a certain endpoint of an arc. It can be any string of characters. However, once specified, the same name must be used in any reference to the object.

arcIndexIdParent, *arcIndexId*, and *nextSerialArcIndexId*
Specifies indexes (integer values) that distinguish between arcs within the same endpoints of the parent arc, the underlying arc, and of the next arc in the series, respectively. It is possible to connect the same two endpoints with several arcs; this parameter provides a means for determining each arc named by the same endpoints.

Note: Avoid duplicate names. There is always a chance of the underlying arc to have exactly the same name as its parent arc, for example if their endpoints are the same and their indexIds are erroneously set to the same value. The NVOT API does check for this, but gtmd does, and it will reject the trap.

ulaIcon
Specifies the symbol to represent this underlying arc in the ovw display. The symbol may be a line, a dotted line, and so on. See the /usr/OV/conf/C/oid_to_sym file for details on how to choose an icon.

ulaLabel
Specifies a string of characters that represent an arc label to be displayed in the drop down menu shown when the right mouse button is clicked on an arc symbol. This is also true for an underlying arc.

Return Values

nvotReturnCode	The nvotCreatSerialUnderlyingArc routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. An arc protocol must be a positive integer and an arc name must not be NULL.
[NVOT_ULA_INVALID_INDEX]	The ULA index is not valid. A ULA protocol must be a positive integer and a ULA name must not be NULL.
[NVOT_ENDPOINT_INVALID_INDEX]	The endpoint index is not valid. An endpoint protocol must be a positive integer and an endpoint name must not be NULL.
[NVOT_ARC_DOES_NOT_EXIST]	The parent arc for which you are creating an underlying arc does not exist in the GTM database.
[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The aEndPoint graph speciified for the underlying arc does not exist in the GTM database.
[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]	The zEndPoint graph speciified for the underlying arc does not exist in the GTM database.

[NVOT_NEXT_SERIAL_ARC_DOES_NOT_EXIST]	The next arc in the series does not exist in the GTM database.
[NVOT_INVALID_NAME_BINDING]	The name binding is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_PROTOCOL_WAS_NOT_REGISTERED]	The protocol was not registered in the /usr/OV/oid_to_protocol file.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

Examples

A printable message string is accessible through a call to the routine nvotGetErrorMsg, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

This example illustrates how to create two arcs in serial underlying of an arc previously created. Notice that the underlying arcs will be created in reverse order. This is a technique to avoid future calls to set the next serial arcs.

```
#include <nvot.h>

nvotReturnCode      RC;
nvotProtocolType    dummyProtocol;
nvotProtocolType    aEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              aEndpointNameParent = "My_Vertex_V1";
nvotProtocolType    zEndpointProtocolParent.vertexProtocol = STARTLAN;
char *              zEndpointNameParent = "My_Vertex_V2";
int                 arcIndexIdParent = 1;
nvotProtocolType    aEndpointProtocol1.vertexProtocol = STARTLAN;
char *              aEndpointName1 = "My_Vertex_V3";
nvotProtocolType    zEndpointProtocol1.vertexProtocol = STARTLAN;
char *              zEndpointName1 = "My_Vertex_V4";
int                 arcIndexId1 = 1;
nvotProtocolType    aEndpointProtocol2.vertexProtocol = STARTLAN;
char *              aEndpointName2 = "My_Vertex_V5";
nvotProtocolType    zEndpointProtocol2.vertexProtocol = STARTLAN;
char *              zEndpointName2 = "My_Vertex_V6";
int                 arcIndexId1 = 1;
char *              icon = "1.3.6.1.2.1.2.2.1.3.54.4";

  RC = nvotCreateSerialUnderlyingArc (
                                  ARC_VERTEX_VERTEX_NAME_BINDING,
                                  aEndpointProtocolParent, aEndpointNameParent,
                                  zEndpointProtocolParent, zEndpointNameParent,
                                  arcIndexIdParent,
```

```
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocol2, aEndpointName2,
                                    zEndpointProtocol2, zEndpointName2,
                                    arcIndexId2, icon, "Ula_V1V2_2",
                                    DONT_CARE_NAME_BINDING,
                                    dummyProtocol, NULL
                                    dummyProtocol, NULL
                                    0);

printf("Create First Serial Ula = %s\n", nvotGetErrorMsg(RC));

RC = nvotCreateSerialUnderlyingArc (
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocolParent, aEndpointNameParent,
                                    zEndpointProtocolParent, zEndpointNameParent,
                                    arcIndexIdParent,
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocol1, aEndpointName1,
                                    zEndpointProtocol1, zEndpointName1,
                                    arcIndexId1, icon, "Ula_V1V2_1",
                                    ARC_VERTEX_VERTEX_NAME_BINDING,
                                    aEndpointProtocol2, aEndpointName2,
                                    zEndpointProtocol2, zEndpointName2,
                                    arcIndexId2);


printf("Create Second Serial Ula = %s\n", nvotGetErrorMsg(RC));
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateParallelUnderlyingArc(3)” on page 240.

nvotCreateUsingSap(3)

Purpose

Creates a SAP of SAP type USING

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotCreateUsingSap (
                       nvotVertexProtocolType  vertexProtocol,
                       char        *           vertexName,
                       nvotVertexProtocolType  sapProtocol,
                       char        *           sapName);
```

Description

Vertices represent communication entities or interfaces across various protocol layers. The SAP object class represents the logical relationship between two vertices inside a computer. If a communication entity in a given protocol layer uses the services of a lower layer entity through a service point, a vertex representing an *N*-layer entity uses a SAP provided by a vertex representing an entity in layer *N-1*. Likewise, a vertex representing an entity in layer *N* can provide a SAP for other vertices representing entities in layer *N+1* to use. Also, an interface or communication entity can provide its services to more than one entity in an upper layer at the same time. However, a SAP always establishes only one association.

This routine correlates a vertex identified by vertexProtocol and vertexName parameter values, using the services of a vertex identified by *sapProtocol* and *sapName*.

Although, in theory, it is possible to multiplex upward and downward, it is not meaningful to create more than one using SAP for a given vertex. This means that the vertex defined by *vertexProtocol* and *vertexName* should be using only one SAP.

Using or providing SAP is a semantical approach for the vertices relationship.

If the vertex using this SAP does not exist in the gtmd database, it is automatically created. However, the automatic creation of vertex requires future calls to the routines nvotChangeVertexIconInBox, nvotChangeVertexLabelInBox, nvotChangeVertexIconInGraph or nvotChangeVertexLabelInGraph for accurate display by the NetView for AIX program.

All parameters in this routine are required.

Parameters

vertexProtocol — Specifies the protocol of the vertex using the SAP. Vertex protocol is an enumerated type defined in the file <nvotTypes.h>.

vertexName — Specifies the name of the vertex using the SAP. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this vertex.

sapProtocol — Specifies the protocol of the vertex in which the SAP is defined. This is the protocol of the vertex providing the SAP. Its value must not be equal to vertexProtocol.

sapName This parameter, or sapAddressName, is the element of correlation. It takes a name that identifies the used SAP. If the SAP used is given by a Token Ring card, for example, this variable is set to the MAC address.

Return Values

nvotReturnCode The nvotCreateUsingSap routine returns an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example established the SNA vertex V1 as using the Token Ring interface.

```
#include <nvot.h>

nvotReturnCode         rc;

/************ Define SNA Session vertex (V1) **************/
nvotVertexProtocolType mySNA_Vert_Prot = SNA_SESSION;
char         *         mySNA_Vert_Name = "USIBMNT.NT67VTAM";

/************ Define Token Ring SAP ***********************/
nvotVertexProtocolType myTokenRingProt = ISO88025_TOKENRING;
char         *         myTokenRingAddr = "10005AA8D718";

  if ((rc =  nvotCreateUsingSap  (mySNA_Vert_Prot,
                                  mySNA_Vert_Name,
                                  myTokenRingProt,
                                  myTokenRingAddr)) == NVOT_SUCCESS);
```

```
    printf ("Sap created successfully");
  else
    printf ("Error : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateProvidingSap(3)” on page 245.
- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateVertexInBox(3)

Purpose

Creates a vertex in a box

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateVertexInBox (
                    nvotGraphProtocolType   boxProtocol,
                    char *                  boxName,
                    nvotVertexProtocolType  vertexProtocol,
                    char *                  vertexName,
                    char *                  icon,
                    char *                  label,
                    nvotOctetString *       vertexDetails,
                    nvotStatusType          status);
```

Description

The nvotCreateVertexInBox routine creates a vertex and associates it with a box graph. Box means a graph with graphType attribute value equal to BOX. A vertex can be a member of several graphs at the same time. If the vertex already exists when nvotCreateVertexInBox is called, the routine creates a new association between the vertex and the box graph. The box graph containing the vertex must exist and its graphType attribute must be set to BOX. Otherwise, the vertex will not be created and an error code will be set.

The protocol and name parameters together uniquely identify objects in the GTM database. These parameters are required.

The parameters icon, label and vertexDetails are optional parameters. If they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is passed, the default **Cards:Generic** symbol is assumed for icon and the vertexName string is displayed in place of label.

The status parameter must be set to one of the values defined in the <nvotTypes.h> file. Otherwise, the routine is rejected and the error NVOT_INVALID_STATUS is set. The status value passed to this routine will be mapped into other NetView for AIX state values according to the table shown in the *NetView for AIX Programmer's Guide*.

Parameters

boxProtocol — Specifies the protocol of the box graph with which this vertex is associated. This is the box graph of which the vertex will be a member. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

boxName — Specifies the name of the box graph with which the vertex is associated. Both the boxName and the boxProtocol are required to identify the parent box graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

vertexProtocol Specifies the protocol of the vertex. The vertexProtocol parameter is an enumerated type defined in the file <nvotTypes.h>.

vertexName Specifies the name of the vertex. It can be any string of characters. Once specified, the same name must be used in any reference to this vertex.

icon Specifies the icon that represents this vertex in the NetView for AIX EUI. Refer to the file /usr/OV/conf/C/oid_to_sym. for details about selecting an icon.

label Specifies the label under the vertex icon in the NetView for AIX EUI. The label parameter can be any string of characters.

vertexDetails Contains particular information that applications store for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(vertexDetails->octetString, (char *) applStruct, sizeof(applStruct)) and vertexDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of vertexDetails, any character exceeding 256 is truncated by the NetView for AIX object database.

status Specifies the status of the vertex. The status parameter is an enumerated type defined in the file <nvotTypes.h>.

Return Values

OVwObjectId When the application is running in synchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), the nvotCreateVertexInBox routine issues the create vertex operation to GTM. The routine remains in a finite loop until the NetView for AIX program returns the OVwObjectId of the vertex just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, the routine returns OVwNullObjectId. When the application is running in asynchronous mode (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), the nvotCreateVertexInBox routine issues the create vertex operation to GTM and immediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the routine nvotGetError. For more information about OVwObjectId, see “nvotSetSynchronousCreation(3)” on page 368.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_OVW_OBJECT_ID_NOT_AVAIL]
An error occurred in creating the object ID for this element.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a vertex of STARLAN protocol inside a box graph. The box graph must already exist. For information about creating a box graph, see the example in “nvotCreateBoxInGraph(3)” on page 227.

```
#include <nvot.h>

OVwObjectId            vertexId;
nvotReturnCode         rc;
nvotBooleanType        synchMode = FALSE;

nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBoxName = "My_Box_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexIcon = "1.3.6.1.2.1.2.2.1.3.11.1";
char         *         myVertexLabel = "My_Star_LAN_Vertex";
nvotOctetString    *   myVertexDetails = NULL;
nvotStatusType         myVertexStatus = STATUS_CRITICAL;
  if (nvotSetSynchronousCreation (TRUE) == NVOT_SUCCESS)
    synchMode = TRUE;

  if ((vertexId = nvotCreateVertexInBox (myBoxProt,
                                         myBoxName,
```

```
                                        myVertexProt,
                                        myVertexName,
                                        myVertexIcon,
                                        myVertexLabel,
                                        myVertexDetails,
                                        myVertexStatus) > OVwNullObjectId)
    printf ("%s OVwObjectId is : %d\n", myVertexLabel, vertexId);
  else
  {
    if (synchMode)
      printf ("An error occurred creating vertex %s\n", myVertexLabel);
  }
  printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotDeleteVertexFromBox(3)” on page 297.
- See “nvotChangeVertexIconInBox(3)” on page 200.
- See “nvotChangeVertexLabelInBox(3)” on page 206.
- See “nvotGetVerticesInBox(3)” on page 353.
- See “nvotSetSynchronousCreation(3)” on page 368.

nvotCreateVertexInGraph(3)

Purpose

Creates a vertex in a graph

Syntax

```
#include <nvot.h>

OVwObjectId     nvotCreateVertexInGraph (
                    nvotGraphProtocolType   graphProtocol,
                    char *                  graphName,
                    nvotVertexProtocolType  vertexProtocol,
                    char *                  vertexName,
                    char *                  icon,
                    char *                  label,
                    nvotOctetString *       vertexDetails,
                    nvotStatusType          status);
```

Description

The nvotCreateVertexInGraph routine creates a vertex and associates it with a graph. A vertex can be a member of several graphs at the same time. If the vertex already exists when nvotCreateVertexInGraph is called, the routine creates a new association between the vertex and a graph. The graph containing the vertex must exist when the routine is called. Otherwise, the vertex will not be created and an error code will be set.

The protocol and name parameters together uniquely identify objects in the GTM database. These parameters are required.

The parameters icon, label and vertexDetails are optional parameters. If they are not passed, they must be set to NULL. Pointers that are not valid might cause unpredictable errors. If NULL is passed, the default **Cards:Generic** symbol is assumed for icon and the vertexName string is displayed in place of label.

The status parameter must be set to one of the values defined in the <nvotTypes.h> file. Otherwise, the routine is rejected and the error [NVOT_INVALID_STATUS] is set. The status value passed to this routine will be mapped into other NetView for AIX state values according to the table shown in the *NetView for AIX Programmer's Guide*.

Parameters

graphProtocol — Specifies the protocol of the graph with which this vertex is associated. This is the protocol of the graph of which this vertex is a member. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the graph with which this vertex is associated. Both the graphName and the graphProtocol are required to identify the parent graph. It can be any string of characters. Once specified, the same name must be used in any reference to this graph.

vertexProtocol — Specifies the protocol of the vertex. The vertexProtocol parameter is an enumerated type defined in the file <nvotTypes.h>.

vertexName Specifies the name of the vertex. It can be any string of characters. Once specified, the same name must be used in any reference to this vertex.

icon Specifies the icon that represents the vertex in the NetView for AIX EUI. Refer to the file /usr/OV/conf/C/oid_to_sym. for details about selecting an icon.

label Specifies the label under the vertex icon in the NetView for AIX EUI. Label can be any string of characters.

vertexDetails Contains particular information that the application stores for future retrieval. The information stored in this variable is for the application's use only. For example, the application might copy the data of a structure into this variable by doing a memcpy(vertexDetails->octetString, (char *) applStruct, sizeof(applStruct)) and vertexDetails->octetLength = sizeof(applStruct). However, although nvotOctetString allows for any size strings and the interface does not check the size of vertexDetails, any character exceeding 256 will be truncated by the NetView for AIX object database.

status Specifies the status of the vertex. The status parameter is an enumerated type defined in the file <nvotTypes.h>.

Return Values

OVwObjectId When the application is running in synchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a non-zero value), the nvotCreateVertexInGraph routine issues the create vertex operation to GTM. The routine stands in a finite loop until the NetView for AIX program returns the OVwObjectId of the vertex just created. OVwObjectId is a positive integer. If an error occurs or the loop times out, the nvotCreateVertexInGraph routine returns OVwNullObjectId. When the application is running in asynchronous mode, (that is, when the nvotSetSynchronousCreation routine has been called with a zero value or has never been called), the routine will issue the create vertex operation to GTM and imediately return OVwNullObjectId. In either case, upon return, an error code is available through a call to the nvotGetError routine. Refer to “nvotSetSynchronousCreation(3)” on page 368 for more details about OVwObjectId.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection went down during operation. Issue the nvotInit routine again.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_OVW_OBJECT_ID_NOT_AVAIL]
An error occurred in creating the object ID for this element.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a vertex of STARLAN protocol inside a graph. The graph must already exist. For more information about creating a graph, see the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

OVwObjectId            vertexId;
nvotReturnCode         rc;
nvotBooleanType        synchMode = FALSE;

nvotGraphProtocolType  my_STARLAN_GraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         my_STARLAN_GraphName = "My_STARLAN_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexIcon = "1.3.6.1.2.1.2.2.1.3.11.1";
char         *         myVertexLabel = "My_STARLAN_Vertex";

nvotOctetString   *    myVertexDetails = NULL;
nvotStatusType         myVertexStatus = STATUS_NORMAL;
  if (nvotSetSynchronousCreation (TRUE) == NVOT_SUCCESS)
    synchMode = TRUE;

  if ((vertexId = nvotCreateVertexInGraph (my_STARLAN_GraphProt,
                                           my_STARLAN_GraphName,
                                           myVertexProt,
                                           myVertexName,
                                           myVertexIcon,
```

```
                                        myVertexLabel,
                                        myVertexDetails,
                                        myVertexStatus) > OVwNullObjectId)
  printf ("%s OVwObjectId is : %d\n", myVertexLabel, vertexId);
else
{
  if (synchMode)
    printf ("An error occurred creating vertex %s\n", myVertexLabel);
}
printf ("Operation result : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

/usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotDeleteVertexFromGraph(3)” on page 299.
- See “nvotChangeVertexIconInGraph(3)” on page 203.
- See “nvotChangeVertexLabelInGraph(3)” on page 209.
- See “nvotChangeVertexPositionInGraph(3)” on page 215.
- See “nvotGetVerticesInGraph(3)” on page 356.
- See “nvotSetSynchronousCreation(3)” on page 368.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotDeleteArc(3)

Purpose

Deletes an arc

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteArc (nvotNameBindingType    arcNameBinding,
                                  nvotProtocolType       aEndpointProtocol,
                                  char        *          aEndpointName,
                                  nvotProtocolType       zEndpointProtocol,
                                  char        *          zEndpointName,
                                  int                    arcIndexId)
```

Description

The nvotDeleteArc routine deletes an arc. An arc connects two arc endpoints: two vertices, two graphs, a vertex to a graph, or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint, and arcIndexId.

The arcNameBinding parameter helps to identify the arc endpoints. See the parameters section of this man page for a detailed description of the arcNameBinding parameter. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters. All parameters are required.

The nvotProtocolType is a union of a enumerated type with a char pointer as defined in the nvotTypes.h file. Special care must be taken when setting aEndpointProtocol and zEndpointProtocol. Setting these variables with a nvotVertexProtocolType value if the arcNameBinding parameter identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

Deleting an arc causes the following side effects:

- The simple connections with which the arc is associated are deleted.
- All underlying arcs belonging to this arc are deleted.
- All graph associations—member arcs and graph-attached arcs— are deleted.

For a detailed explanation of the side effects of deleting an arc, see the chapter that discusses NetView for AIX open topology side effects in the *NetView for AIX Programmer's Guide*.

Parameters

arcNameBinding
: Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

  ARC_VERTEX_VERTEX_NAME_BINDING
  : Indicates that either endpoint is a vertex

  ARC_VERTEX_GRAPH_NAME_BINDING
  : Indicates that aEndpoint is a vertex and zEndpoint is a graph

  ARC_GRAPH_VERTEX_NAME_BINDING
  : Indicates that aEndpoint is a graph and zEndpoint is a vertex

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates that either endpoint is a graph

If any value other than the preceding values is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs can be handled based on their direction. For more information about the direction of arcs, see “nvotInit(3)” on page 359. Regardless of which direction was set in the nvotInit routine, the arcNameBinding parameter always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. If aEndpoint or zEndpoint is to be a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set with a value from the enumerated type nvotVertexProtocolType, which is defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName/zEndPointName
Specifies the name of the object identified as the aEndpoint or zEndpoint, respectively, of this arc. Both the endpoint name and the endpoint protocol are required to identify the object at one of the endpoints of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

Return Values

nvotReturnCode The nvotDeleteArc routine returns an nvotReturnCode that can assume the values described in the error codes section of this man page.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes arc number 1 connecting oneEndpoint to otherEndpoint.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;
int                    arcNumber = 1;
  if ((rc =  nvotDeleteArc (ARC_VERTEX_VERTEX_NAME_BINDING,
                            oneEndpoint,
                            oneEndpointName,
                            otherEndpoint,
                            otherEndpointName,
                            arcNumber)) == NVOT_SUCCESS)

    printf ("Arc from %s to %s of index %d deleted.\n",
                                  oneEndpointName, otherEndpointName, arcNumber);
  else
    printf ("An error occurred deleting arc number %d from %s to %s.\n",
                                  oneEndpointName, otherEndpointName, arcNumber);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.
- See “nvotDeleteArcFromGraph(3)” on page 272.
- See “nvotGetArcsInGraph(3)” on page 307.

nvotDeleteArcFromGraph(3)

Purpose

Deletes an arc from a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteArcFromGraph (
                       nvotGraphProtocolType  graphProtocol,
                       char *                 graphName,
                       nvotNameBindingType    arcNameBinding,
                       nvotProtocolType       aEndpointProtocol,
                       char *                 aEndpointName,
                       nvotProtocolType       zEndpointProtocol,
                       char *                 zEndpointName,
                       int                    arcIndexId);
```

Description

The nvotDeleteArcFromGraph routine deletes the relationship that associates the arc identified by aEndpoint, zEndpoint and arcIndexId from the graph identified by graphProtocol and graphName. Unlike the nvotDeleteArc routine, this routine deletes the arc only if it is not a member of any graph.

The deletion of an arc relationship causes no side effects. However, if an arc is deleted, there are several side affects. These side affects are described in “nvotDeleteArc(3)” on page 269.

An arc connects arc endpoints: two vertices, two graphs, a vertex to a graph, or a graph to a vertex. An arc is recognized and referenced by its aEndpoint, zEndpoint, and arcIndexId.

The arcNameBinding parameter helps to identify the arc endpoints. For a detailed description, see the following parameters section in this man page. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters. All parameters are required.

The nvotProtocolType is a union of a enumerated type with a char pointer as defined in nvotTypes.h file. Special care must be taken when setting the aEndpointProtocol and zEndpointProtocol parameters. Setting these variables to an nvotVertexProtocolType value if arcNameBinding identifies the endpoint as a graph causes unpredictable errors. This is similar to setting a char pointer to an integer value.

Parameters

graphProtocol
Specifies the protocol of the graph that contains the arc. This is the graph of which this arc is a member arc. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName
Specifies the name of the graph that contains the arc. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter is a string of characters used to create the graph.

arcNameBinding
Specifies the class of the objects in each endpoint of the arc. An endpoint can be either a vertex or a graph. The following values are supported:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates that either endpoint is a vertex.

ARC_VERTEX_GRAPH_NAME_BINDING
Indicates that aEndpoint is a vertex and zEndpoint is a graph

ARC_GRAPH_VERTEX_NAME_BINDING
Indicates that aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either endpoints are graphs

If a value other than those in the preceding list is used, it is rejected by the GTM interface and the error code NVOT_INVALID_NAME_BIND is set.

Arcs are handled based on their direction. For more information about arc direction, see “nvotInit(3)” on page 359. Regardless of the selection made in the nvotInit routine, arcNameBinding always identifies what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol/zEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint or aEndpointProtocol, respectively, of this arc. If aEndpoint or zEndpoint is a vertex, aEndpointProtocol or zEndpointProtocol, respectively, must be set to a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, aEndpoint or zEndpoint is a graph, and aEndpointProtocol or zEndpointProtocol, respectively, is a pointer to a valid character string in memory.

aEndpointName
Specifies the name of the object identified as the aEndpoint of this arc. Both the aEndpointName and aEndpointProtocol parameters are required to identify the object at the aEndpoint of this arc. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

arcIndexId
Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arcIndexId is an integer value.

Return Values

nvotReturnCode The nvotDeleteArcFromGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the dotted arc created in the example in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Graph";
nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_Endpoint";
nvotProtocolType       otherEndpoint.graphProtocol = "1.3.6.1.2.1.2.2.1.3.11";
char         *         otherEndpointName = "My_Graph_Endpoint";
int                    arcNumber = 1;
char         *         myDotDashArcLabel = "My_Dotted_Arc"

  if (rc =  nvotDeleteArcFromGraph (myGraphProt,
                                    myGraphName,
                                    ARC_VERTEX_GRAPH_NAME_BINDING,
                                    oneEndpoint,
                                    oneEndpointName,
                                    otherEndpoint,
                                    otherEndpointName,
                                    arcNumber) == NVOT_SUCCESS)

    printf ("%s deleted successfully.\n", myDotDashArcLabel);
  else
    printf ("An error occurred deleting arc %s\n", myDotDashArcLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.
- See “nvotGetArcsInGraph(3)” on page 307.

nvotDeleteBox(3)

Purpose

Deletes a box

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteBox (nvotGraphProtocolType  boxProtocol,
                                  char         *         boxName);
```

Description

The nvotDeleteBox routine deletes a box graph. The protocol and name parameters uniquely identify objects in the GTM database. Both parameters are required.

Deleting a box causes side effects, including the following:

- Additional information about the box graph and box graph members is deleted.
- All associations with vertices and arcs, including members, member arcs and graph-attached arcs, are deleted.
- All arcs and simple connections named by this graph are deleted. For a detailed explanation of the effects of deleting a graph, refer to the chapter discussing NetView for AIX open topology side effects in the *NetView for AIX Programmer's Guide*.

Parameters

boxProtocol Specifies the protocol of the box graph. For more information about specifying a box protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the box graph. Both the boxName and boxProtocol parameters are required to uniquely identify the box graph. This parameter is a string of characters previously used to create the box graph.

Return Values

nvotReturnCode The nvotDeleteBox routine returns an nvotReturnCode that can assume the values described in the error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following sample deletes a box graph.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myBox_STARLAN_GraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";

  if ((rc =  nvotDeleteBox (myBox_STARLAN_GraphProt,
                            myBox_STARLAN_GraphName)) == NVOT_SUCCESS)

    printf ("%s deleted.\n", myBox_STARLAN_GraphName);
  else
    printf ("An error occurred deleting box : %s\n", myBox_STARLAN_GraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotDeleteBoxFromGraph(3)” on page 278.
- See “nvotGetBoxesInGraph(3)” on page 314.

nvotDeleteBoxFromGraph(3)

Purpose

Deletes a box from a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteBoxFromGraph (
                       nvotGraphProtocolType  graphProtocolParent,
                       char *                 graphNameParent,
                       nvotGraphProtocolType  boxProtocol,
                       char *                 boxName);
```

Description

The nvotDeleteBoxFromGraph routine deletes the relationship that associates a box graph identified by boxProtocol and boxName to a parent graph identified by graphProtocolParent and graphNameParent.

This routine deletes a box graph only if it is not a member of any other graph and it contains no members, memberArcs and attachedArcs in it.

The deletion of the box graph relationship causes no side effects. However, if a box graph is deleted, there are several side effects. These side effects are described in “nvotDeleteGraph(3)” on page 281.

The protocol and name parameters uniquely identify objects in the GTM database. These parameters are required for both the parent and box graphs.

Parameters

graphProtocolParent Specifies the protocol of the parent graph. For more information about specifying a graph's protocol refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph.

boxProtocol Specifies the protocol of the child box graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the child box graph. Both the boxName and the boxProtocol parameters are required to identify the box graph.

Return Values

nvotReturnCode The nvotDeleteBoxFromGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol and/or name must not be NULL.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box graph protocol and/or name must not be NULL.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following sample deletes a box that is a member of the root graph.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";

char         *         myBox_STARLAN_GraphName = "My_Box_STARLAN_Graph";
char         *         myBox_STARLAN_GraphLabel = "My_Box_STARLAN_Graph"

  if ((rc = nvotDeleteBoxFromGraph (my_STARLAN_GraphsProt,
                                    myRoot_STARLAN_GraphName,
                                    my_STARLAN_GraphsProt,
                                    myBox_STARLAN_GraphName)) == NVOT_SUCCESS)

    printf ("%s deleted successfully.\n", myBox_STARLAN_GraphLabel);
  else
    printf ("An error occurred deleting box %s\n", myBox_STARLAN_GraphLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotGetBoxesInGraph(3)” on page 314.

nvotDeleteGraph(3)

Purpose

Deletes a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteGraph (nvotGraphProtocolType  graphProtocol,
                                    char       *           graphName)
```

Description

The nvotDeleteGraph routine deletes a graph. The protocol and name parameters together uniquely identify objects in the GTM database. Both parameters are required.

Deleting a graph causes side effects, including the following:

- Additional information about the graph and graph members is deleted.
- All associations with vertices and arcs, including members, member arcs and graph-attached arcs, are deleted.
- All arcs and simple connections named by this graph are deleted.

For a detailed explanation of the effects of deleting a graph, refer to the chapter discussing NetView for AIX open topology side effects in the *NetView for AIX Programmer's Guide*.

Parameters

graphProtocol Specifies the protocol of the graph. For more information about specifying a graph protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the graph. Both the graphName and the graphProtocol are required to uniquely identify the graph in the GTM database. This parameter is a string of characters previously used to create the graph.

Return Values

nvotReturnCode The nvotDeleteGraph routine returns an nvotReturnCode that can assume the values described in the error codes section in this man page.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the graph created in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         mySDLCGraphName = "My_Child_SDLC_Graph";

  if ((rc =  nvotDeleteGraph (mySDLCGraphProt,
                              mySDLCGraphName)) == NVOT_SUCCESS)

    printf ("%s deleted.\n", mySDLCGraphName);
  else
    printf ("An error occurred deleting graph : %s\n", mySDLCGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotDeleteGraphFromGraph(3)” on page 283.
- See “nvotGetGraphsInGraph(3)” on page 330.

nvotDeleteGraphFromGraph(3)

Purpose

Deletes a graph from a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteGraphFromGraph (
                       nvotGraphProtocolType  graphProtocolParent,
                       char *                 graphNameParent,
                       nvotGraphProtocolType  graphProtocol,
                       char *                 graphName);
```

Description

The nvotDeleteGraphFromGraph routine deletes the relationship that associates a child graph identified by graphProtocol and graphName to a parent graph identified by graphProtocolParent and graphNameParent.

The child graph is deleted only if it is not a member of any other graph and it contains no members, memberArcs and attachedArcs.

The deletion of the graph relationship causes no side effects. However, if the child graph is deleted, there are several side effects. These side effects are described in “nvotDeleteGraph(3)” on page 281.

The protocol and name parameters uniquely identify objects in the GTM database. These parameters are required for both the parent and child graphs.

Parameters

graphProtocolParent Specifies the protocol of the parent graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the parent graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph.

graphProtocol Specifies the protocol of the child graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the child graph. Both the graphName and graphProtocol parameters are required to uniquely identify the child graph.

Return Values

nvotReturnCode The nvotDeleteGraphFromGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol and/or name must not be NULL.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes a graph, member of a root graph, created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myChildSDLCGraphName = "My_Child_SDLC_Graph";
char         *         myChildSDLCGraphLabel = "My_Child_SDLC_Graph";

  if ((rc = nvotDeleteGraphFromGraph (mySDLCGraphsProt,
                                      myRootSDLCGraphName,
                                      mySDLCGraphsProt,
                                      myChildSDLCGraphName)) == NVOT_SUCCESS)

    printf ("%s deleted successfully.\n", myChildSDLCGraphLabel);
  else
    printf ("An error occurred deleting graph %s\n", myChildSDLCGraphLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotGetGraphsInGraph(3)” on page 330.

nvotDeleteProvidingSap(3)

Purpose

Deletes a SAP of SAP type PROVIDING

Syntax

```
#include <nvot.h>

nvotReturnCode nvotDeleteProvidingSap (nvotVertexProtocolType vertexProtocol,
                                       char        *          vertexName,
                                       nvotVertexProtocolType sapProtocol,
                                       char        *          sapName)
```

Description

This nvotDeleteProvidingSap routine deletes a SAP from the list of SAPs provided by the vertex identified by vertexProtocol and vertexName. The sapProtocol and sapName parameters, as well as the vertexProtocol and vertexName parameters of the vertex providing the SAP, identify the SAP to be deleted. All of these parameters are required. If one of these parameters is not provided, the error code NVOT_VERTEX_INVALID_INDEX or NVOT_SAP_INVALID_INDEX is returned.

A SAP exists in vertex V1 to provide services to vertex V2 using it. A given SAP can be referenced by vertex V1, which provides it, and vertex V2, which uses it, at the same time. In this case, a call to the nvotDeleteProvidingSap routine does not delete the SAP itself but removes the reference to the SAP from the list of SAPs provided by a vertex. A further call to the nvotDeleteUsingSap routine deletes the SAP itself. See “nvotDeleteUsingSap(3)” on page 292.

Parameters

vertexProtocol Specifies the protocol of the vertex providing the SAP. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex. This parameter is a string of characters used to create the vertex.

sapProtocol Specifies the protocol of the vertex in which the SAP is defined. This is the protocol of the vertex providing this SAP. See the following example in this man page.

sapName The sapName or sapAddressName parameter is used to identify a SAP provided by an *N*-level entity to an *N+1*-level entity. The sapName parameter is a character string containing information including an IP address, and an SNA physical and logical unit address. See the following example.

Return Values

nvotReturnCode The nvotDeleteProvidingSap routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example bellow deletes a SAP which has been created in the example given in routine “nvotCreateProvidingSap(3)” on page 245.

```
#include <nvot.h>

nvotReturnCode         rc;

/************ Define Token Ring vertex (V1) ***************/
nvotVertexProtocolType myTokenRingProt = ISO88025_TOKENRING;
char         *         myTokenRingName = "TR_Card";

/************ Define TCP/IP vertex (V3) ******************/
nvotVertexProtocolType myTCP_IP_Prot   = IP;
char         *         myTCP_IP_Name   = "9.179.1.237";

  if ((rc =  nvotDeleteProvidingSap (myTokenRingProt,
                                     myTokenRingName,
                                     myTokenRingProt,
                                     myTCP_IP_Name)) == NVOT_SUCCESS)

    printf ("Sap provided to TCP/IP has been deleted.\n");
  else
    printf ("An error occurred deleting sap.\n");
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateUsingSap(3)” on page 258.
- See “nvotCreateProvidingSap(3)” on page 245.

nvotDeleteUnderlyingArc(3)

Purpose

Deletes an underlying arc relationship to its parent arc.

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteUnderlyingArc  (
                         nvotNameBindingType        arcNameBindingParent,
                         nvotProtocolType           aEndpointProtocolParent,
                         char *                     aEndpointNameParent,
                         int                        arcIndexIdParent,
                         nvotNameBindingType        arcNameBinding,
                         nvotProtocolType           zEndpointProtocol,
                         char *                     aEndpointName,
                         nvotProtocolType           zEndpointProtocol,
                         char *                     zEndpointName,
                         int                        arcIndexId)
```

Description

This routine deletes an underlying arc relationship to its parent arc. It does not delete the child arc itself. See “nvotDeleteArc(3)” on page 269 for information about deleting the arc. If this routine completes successfully, the underlying arc symbol is removed from the parent arc submap.

Parameters

arcNameBindingParent and *arcNamebinding*
Specify the class of the objects in each endpoint of the parent arc and the underlying arc itself, respectively. The endpoint could be either a vertex or a graph. The allowed values are as follows:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates either of the endpoints are vertices.

ARC_VERTEX_GRAPH_NAME_BINDING
aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either of the endpoints are graphs.

aEndpointProtocolParent, zEndpointProtocolParent, aEndpointProtocol, and *zEndpointProtocol*
These parameters are all of the same type. They specify the protocol of the object identified as the endpoint of the parent arc and the underlying arc itself, respectively. For instance, if aEndpoint is to be a vertex, aEndpointProtocol must be set to a value out of the enumerated type nvotVertexProtocolType defined in the file <nvotTypes.h>. Otherwise, aEndpoitn is a agraph, and aEndpointProtocol must take a pointer to a valid character string in memory.

aEndpointNameParent, zEndpointNameParent, aEndpointName, and *zEndpointName*
These parameters are all of type char *. They specify the name of the object identified as the endpoint of the parent arc and the underlying arc, respectively. The endpointName, together with the endpointProtocol, are required to identify the object at a certain endpoint of an arc. It must be the very same string of characters used in the creation of the underlying arc.

arcIndexIdParent and *arcIndexId*
Since it is possible to connect the same two endpoints with several arcs, there should be a way to distinguish each arc named by the same endpoints. These indexes are integer values that distinguish one arc among others between the same endpoints of the parent arc and the underlying arc, respectively.

Return Values

nvotReturnCode The nvotDeleteUnderlyingArc routine returns an nvotReturnCode that can assume the values described in the following error codes.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. An arc index must be a positive integer.
[NVOT_ULA_INVALID_INDEX]	An underlying arc index is not valid.
[NVOT_INVALID_NAME_BINDING]	Invalid name binding. The name must be a number defined in the file <nvotTypes.h>.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example shows one of the underlying arcs created in the example given in the routine “nvotCreateParallelUnderlyingArc(3)” on page 240.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotProtocolType       oneEndpoint.VertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.VertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2";
int                    parentArc = 1;
int                    oneUla    = 2;

  RC = nvotDeleteUnderlyingArc (ARC_VERTEX_NAME_BINDING,
                                oneEndpoint, oneEndpointName,
                                otherEndpoint, otherEndpointName,
                                parentArc,
                                ARC_VERTEX_VERTEX_NAME_BINDING,
                                oneEndpoint, oneEndpointName,
                                otherEndpoint, otherEndpointName,
                                oneUla);

  printf ("DeleteUla=  %s\n", nvotGetErrorMsg (RC));
  print  ("Ula_V1V2_2\n\n");
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateParallelUnderlyingArc(3)” on page 240.
- See “nvotCreateSerialUnderlyingArc(3)” on page 253.
- See “nvotDeleteArc(3)” on page 269.

nvotDeleteUsingSap(3)

Purpose

Deletes a SAP of SAP type USING

Syntax

```
#include <nvot.h>

nvotReturnCode nvotDeleteUsingSap (nvotVertexProtocolType vertexProtocol,
                                   char        *          vertexName,
                                   nvotVertexProtocolType sapProtocol,
                                   char        *          sapName)
```

Description

The nvotDeleteUsingSap routine deletes a SAP from the list of SAPs used by the vertex identified by vertexProtocol and vertexName. The sapProtocol and sapName parameters, as well as the vertexProtocol and vertex Name parameters of the vertex using the SAP, identify the SAP to be deleted. All these parameters are required. If one of these parameters is not provided, the error code [NVOT_VERTEX_INVALID_INDEX] or [NVOT_SAP_INVALID_INDEX] is returned.

A using SAP exists for vertex V1 because there is a second vertex, V2, providing it. A given SAP can be referenced by vertex V2, which provides it, and vertex V1, which uses it, at the same time. In this case, a call to the nvotDeleteUsingSap routine does not delete the SAP itself but removes the reference to the SAP from the list of SAPs used by a vertex. A further call to the nvotDeleteProvidingSap routine deletes the SAP itself. See “nvotDeleteProvidingSap(3)” on page 286 for more information.

Parameters

vertexProtocol — Specifies the protocol of the vertex using the SAP. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex. This parameter is a string of characters used to create the vertex.

sapProtocol — Specifies the protocol of the vertex in which the SAP is defined. This is the protocol of the vertex providing the SAP. See the following example.

sapName — The sapName or sapAddressName parameter is used to identify a SAP provided by an *N*-level entity to an *N+1*-level entity. The sapName parameter is a character string containing information including an IP address, and a SNA physical and logical unit address. See the following example.

Return Values

nvotReturnCode — The nvotDeleteUsingSap routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] — Successful operation.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the SAP that is created by the code in the example in “nvotCreateUsingSap(3)” on page 258.

```
#include <nvot.h>

nvotReturnCode         rc;

/************ Define Token Ring vertex (V1) ***************/
nvotVertexProtocolType myTokenRingProt = ISO88025_TOKENRING;
char         *         myTokenRingName = "TR_Card";

/************ Define TCP/IP vertex (V3) ******************/
nvotVertexProtocolType myTCP_IP_Prot   = IP;
char         *         myTCP_IP_Name   = "9.179.1.237";

  if ((rc =  nvotDeleteUsingSap (myTCP_IP_Prot,
                                 myTCP_IP_Name,
                                 myTokenRingProt,
                                 myTCP_IP_Name)) == NVOT_SUCCESS)

    printf ("LLC Sap in use by TCP/IP has been deleted.\n");
  else
    printf ("An error occurred deleting sap.\n");
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateUsingSap(3)” on page 258.
- See “nvotCreateProvidingSap(3)” on page 245.
- See “nvotDeleteProvidingSap(3)” on page 286.

nvotDeleteVertex(3)

Purpose

Deletes a vertex

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteVertex (nvotVertexProtocolType  vertexProtocol,
                                     char         *          vertexName)
```

Description

The nvotDeleteVertex routine deletes from the GTM database a vertex identified by the vertexProtocol and vertexName parameters. These parameters are required.

When you use the nvotDeleteVertex routine to delete a vertex, the following actions also occur:

- All arcs that have this vertex as endpoint are deleted.
- All SAPs provided or used by the vertex are deleted.
- The vertex's membership in any graphs with which it is associated is discontinued.

The deletion of arcs or SAPs, or the discontinuation of a vertex's membership in graphs can cause further deletions. For example, when these arcs are deleted, underlying arcs, graph-attached arcs, and simple connections associated with these arcs can also be deleted. For a complete explanation of the effects of deleting a vertex, see the chapter that discusses NetView for AIX open topology side effects in the *NetView for AIX Programmer's Guide*.

Parameters

vertexProtocol Specifies the protocol of the vertex to be deleted. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex to be deleted. This parameter can be any string of characters that, in conjunction with vertexProtocol, identifies a vertex in the GTM database.

Return Values

nvotReturnCode The nvotDeleteVertex routine returns an nvotReturnCode that can assume the values described in the error codes section of this man page.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the vertex created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";

  if ((rc =  nvotDeleteVertex (myVertexProt,
                               myVertexName)) == NVOT_SUCCESS)

    printf ("Vertex %s deleted.\n", myVertexName);
  else
    printf ("An error occurred deleting vertex : %s\n", myVertexName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotCreateVertexInBox(3)” on page 261.
- See “nvotDeleteVertexFromGraph(3)” on page 299.
- See “nvotDeleteVertexFromBox(3)” on page 297.
- See “nvotGetVerticesInGraph(3)” on page 356.
- See “nvotGetVerticesInBox(3)” on page 353.

nvotDeleteVertexFromBox(3)

Purpose

Deletes a vertex from a box

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteVertexFromBox (
                       nvotGraphProtocolType   boxProtocol,
                       char *                  boxName,
                       nvotVertexProtocolType  vertexProtocol,
                       char *                  vertexName);
```

Description

The nvotDeleteVertexFromBox routine deletes the relationship that associates a vertex identified by vertexProtocol and vertexName to a box graph identified by boxProtocol and boxName. Unlike the nvotDeleteVertex routine, this routine deletes a vertex only if it is not a member of any other graph.

The deletion of a graph's vertex relationship causes no side effects. However, if a vertex is deleted, there are several side effects. These are described in “nvotDeleteVertex(3)” on page 295.

All parameters are required.

Parameters

boxProtocol — Specifies the protocol of the box graph that contains the vertex. This is the box graph of which this vertex is a member. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

boxName — Specifies the name of the box graph that contains the vertex. Both the boxName and boxProtocol parameters are required to identify the containing box graph. This parameter is a string of characters used to create the box.

vertexProtocol — Specifies the protocol of the vertex associated with the box graph. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex associated with the box graph. This parameter is a string of characters used to create the vertex.

Return Values

nvotReturnCode — The nvotDeleteVertexFromBox routine returns an nvotReturnCode that can assume the values described in the following error codes section in this man page.

Error Codes

[NVOT_SUCCESS] — Successful operation.

[NVOT_BOX_INVALID_INDEX] — The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the vertex created in “nvotCreateVertexInBox(3)” on page 261.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBoxName = "My_Box_Graph";
nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexLabel = "My_Star_LAN_Vertex";

  if (rc = nvotDeleteVertexFromBox (myBoxProt,
                                    myBoxName,
                                    myVertexProt,
                                    myVertexName) == NVOT_SUCCESS)

    printf ("%s deleted successfully.\n", myVertexLabel);
  else
    printf ("An error occurred deleting vertex %s\n", myVertexLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInBox(3)” on page 261.
- See “nvotChangeVertexIconInBox(3)” on page 200.
- See “nvotChangeVertexLabelInBox(3)” on page 206.
- See “nvotGetVerticesInBox(3)” on page 353.

nvotDeleteVertexFromGraph(3)

Purpose

Deletes a vertex from a graph

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotDeleteVertexFromGraph (
                       nvotGraphProtocolType   graphProtocol,
                       char *                  graphName,
                       nvotVertexProtocolType  vertexProtocol,
                       char *                  vertexName);
```

Description

The nvotDeleteVertexFromGraph routine deletes the relationship that associates a vertex identified by vertexProtocol and vertexName to a graph identified by graphProtocol and graphName. Unlike the nvotDeleteVertex routine, this routine deletes a vertex only if it is not a member of any other graph.

The deletion of a graph's vertex relationship causes no side effects. However, if a vertex is deleted, there are several side effects. These are described in “nvotDeleteVertex(3)” on page 295.

All parameters are required.

Parameters

graphProtocol Specifies the protocol of the graph that contains the vertex. This is the graph of which the vertex is a member. For more information, refer to the file </usr/OV/conf/oid_to_protocol>.

graphName Specifies the name of the graph that contains the vertex. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter is a string of characters used to create the graph.

vertexProtocol Specifies the protocol of the vertex associated with the graph. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName Specifies the name of the vertex associated with the graph. This parameter is a string of characters used to create the vertex.

Return Values

nvotReturnCode The nvotDeleteVertexFromGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section in this man page.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol and/or name must not be NULL.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes the vertex created in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         my_STARLAN_GraphName = "My_STARLAN_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
char         *         myVertexLabel = "My_STARLAN_Vertex";

  if (rc = nvotDeleteVertexFromGraph (my_STARLAN_GraphProt,
                                      my_STARLAN_GraphName,
                                      myVertexProt,
                                      myVertexName) == NVOT_SUCCESS)

    printf ("%s deleted successfully.\n", myVertexLabel);
  else
    printf ("An error occurred deleting vertex %s\n", myVertexLabel);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotChangeVertexIconInBox(3)” on page 200.
- See “nvotChangeVertexLabelInGraph(3)” on page 209.
- See “nvotChangeVertexPositionInGraph(3)” on page 215.
- See “nvotGetVerticesInGraph(3)” on page 356.

nvotDone(3)

Purpose

Closes connections and terminates interface activity

Syntax

```
#include <nvot.h>
nvotReturnCode  nvotDone ()
```

Description

The nvotDone routine closes the connection to gtmd. It also closes the connection to the NetView for AIX program if nvotSetSynchronousCreation (TRUE) has been called.

If the interface has not been initialized or the connections have not been closed, the nvotDone routine does not take any action and the error code NVOT_NOT_INITIALIZED is returned.

Return Values

If successful, nvotDone returns [NVOT_SUCCESS]. If unsuccessful, nvotDone returns one of the following error codes.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS]	Successful operation.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

A printable message string is accessible through a call to the routine nvotGetErrorMsg, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example closes a connection previously established to gtmd and checks the result.

```
#include <nvot.h>

nvotReturnCode         rc;

  if ((rc = nvotDone ()) == NVOT_SUCCESS)
    printf ("OK : %s\n", nvotGetErrorMsg (rc));
  else
    printf ("WHOOPS! : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotInit(3)” on page 359.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotFree(3)

Purpose

Releases memory allocated by routines in the GTM API

Related Functions

- nvotFreeVertex
- nvotFreeVertexList
- nvotFreeGraph
- nvotFreeGraphList
- nvotFreeBox
- nvotFreeBoxList
- nvotFreeArc
- nvotFreeArcList
- nvotFreeSap
- nvotFreeSapList
- nvotFreeSimpleConnection
- nvotFreeSimpleConnectionList
- nvotFreeUnderlyingConnection
- nvotFreeUnderlyingConnectionList
- nvotFreeUnderlyingArc
- nvotFreeUnderlyingArcList
- nvotFreeMembers
- nvotFreeMembersList
- nvotFreeMemberArcs
- nvotFreeMemberArcsList
- nvotFreeAttachedArcs
- nvotFreeAttachedArcsList
- nvotFreeAdditionalMembers
- nvotFreeAdditionalMembersList
- nvotFreeAdditionalGraph
- nvotFreeAdditionalGraphList

Syntax

```
#include <nvot.h>

nvotReturnCode  nvotFreeVertex (nvotVertex * vertex)
nvotReturnCode  nvotFreeVertexList (nvotVertexList * vertexList)

nvotReturnCode  nvotFreeGraph (nvotGraph * graph)
nvotReturnCode  nvotFreeGraphList (nvotGraphList * graphList)

nvotReturnCode  nvotFreeBox (nvotBox * box)
nvotReturnCode  nvotFreeBoxList (nvotBoxList * boxList)

nvotReturnCode  nvotFreeArc (nvotArc * arc)
nvotReturnCode  nvotFreeArcList (nvotArcList * arcList)

nvotReturnCode  nvotFreeSap (nvotSap * sap)
nvotReturnCode  nvotFreeSapList (nvotSapList * sapList)

nvotReturnCode  nvotFreeSimpleConnection (nvotSimpleConnection * simpleConnection)
nvotReturnCode  nvotFreeSimpleConnectionList (nvotSimpleConnectionList * simpleConnectionList)

nvotReturnCode  nvotFreeUnderlyingConnection (nvotUnderlyingConnection * underlyingConnection)
nvotReturnCode  nvotFreeUnderlyingConnectionList (nvotUnderlyingConnectionList * underlyingConnectionList)

nvotReturnCode  nvotFreeUnderlyingArc (nvotUnderlyingArc * underlyingArc)
nvotReturnCode  nvotFreeUnderlyingArcList (nvotUnderlyingArcList * underlyingArcList)

nvotReturnCode  nvotFreeMembers (nvotMembers * members)
nvotReturnCode  nvotFreeMembersList (nvotMembersList * membersList)

nvotReturnCode  nvotFreeMemberArcs (nvotMemberArcs * memberArcs)
nvotReturnCode  nvotFreeMemberArcsList (nvotMemberArcsList * memberArcsList)

nvotReturnCode  nvotFreeAttachedArcs (nvotAttachedArcs * attachedArcs)
nvotReturnCode  nvotFreeAttachedArcsList (nvotAttachedArcsList * attachedArcsList)

nvotReturnCode  nvotFreeAdditionalMembers (nvotAdditionalMembers * additionalMembers)
nvotReturnCode  nvotFreeAdditionalMembersList (nvotAdditionalMembersList * additionalMembersList)

nvotReturnCode  nvotFreeAdditionalGraph (nvotAdditionalGraph * additionalGraph)
nvotReturnCode  nvotFreeAdditionalGraphList (nvotAdditionalGraphList * additionalGraphList)
```

Description

The get routines return structures or lists of structures to the application. These structures and lists are memory that have been allocated by the interface.

The free routines are available for the application to release the memory allocated by the get routines.

Parameters

For the definitions of the structures and lists, see Chapter 6, “Using NetView for AIX GTM Data Structures” on page 1061.

Return Values

nvotReturnCode The routine returns NVOT_SUCCESS.

Error Codes

[NVOT_SUCCESS]	The operation was successful.

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

nvotGetArcsInGraph(3)

Purpose

Gets a list of all arcs contained in a graph

Syntax

```
#include <nvot.h>

nvotArcList   *   nvotGetArcsInGraph (
                      nvotGraphProtocolType   graphProtocol,
                      char          *         graphName)
```

Description

The nvotGetArcsInGraph routine issues a get operation of all arcs contained in the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the routine does not search for an arc list and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the containing graph does not exist, the routine does not search for arcs and the error code NVOT_GRAPH_DOES_NOT_EXIST is set.

If the get operation fails, the routine returns NULL and the error variable is set. See the following error codes and return values sections.

If the get operation completes successfully but no arc exists within the graph, the routine returns NULL and the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and arcs exist in the graph, the routine returns a list of all arcs contained in the identified graph.

Notice that the interface allocates structured data in memory and returns a pointer to it. The user must call one of the free memory routines to have all data de-allocated.

Parameters

graphProtocol Specifies the protocol of the graph this routine looks at. This is the graph for whose arcs the routine searches. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the graph this routine looks at. Both the graphName and graphProtocol parameters are required to uniquely identify the containing graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

Return Values

nvotArcList

Upon completion of the get operation, the nvotGetArcsInGraph routine returns a list of all arcs that are members of the identified graph.

As defined in the file nvotTypes.h, nvotArcList is an array list of nvotArc structures. The nvotArcList return value is a structure made up of a pointer to the first nvotArc element and an integer variable count indicating the number of elements in the list.

Each nvotArc element is a structure carrying information about an arc. For more information, please see “Basic Structures” on page 1061.

The variable operation returned in each nvotArc element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all arcs contained in the graph created in the example shown in “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotArcList     *      myArcs = NULL;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Graph";

  if ((myArcs = nvotGetArcsInGraph (myGraphProt,
                                    myGraphName) != NULL) {

    /* OK, it seems we have gotten a few arcs. Print their names. */
    printf ("Graph %s contains %d arcs.\n", myGraphName, myArcs->count);
    for (i = 1; i = myArcs->count; i++)
      printf ("Arc %d = %s<->%s, #%d.\n", i, myArcs->arc[i-1].arcAttr.aEndpointName,
                                             myArcs->arc[i-1].arcAttr.zEndpointName,
                                             myArcs->arc[i-1].arcAttr.arcIndexId);

    /* We don't need them any longer. Let's release all memory. */
    nvotFreeArcList (myArcs);

  } else {
    /* No, we've gotten no arcs. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("Graph %s contains no arcs.\n", myGraphName);
    else {
      printf ("Error occurred getting arcs from graph %s\n", myGraphName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See nvotFreeArc in “nvotFree(3)” on page 304.
- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetArcObjectId(3)

Purpose

Gets an arc ObjectId from the OVW database

Syntax

```
#include <nvot.h>

OVwObjectId   nvotGetArcObjectId (
                      nvotNameBindingType     arcNameBinding,
                      nvotProtocolType        aEndpointProtocol,
                      char     *              aEndpointName,
                      nvotProtocolType        zEndpointProtocol,
                      char     *              zEndpointName,
                      int                     arcIndexId)
```

Description

Objects in the OVW database are given object IDs. When an object is created in its database, gtmd sends a notification to ovwdb, which creates an ObjectId in its own database. These actions are not synchronous and the time elapsed between GTM creation and OVW creation might vary, depending on the activity of GTM and OVW. This is a consideration for applications that directly connect to the OVW database with the intention of adding new variables to that database.

This routine returns the OVwObjectId of the arc identified by aEndpointProtocol/Name, zEndpointProtocol/Name and arcIndexId.

An arc might exist to connect two vertices, two graphs or a vertex to a graph and vice versa. These are called arc endpoints. An arc is recognized and referenced by its aEndpoint, zEndpoint and arcIndexId. Thus, aEndpointProtocol, aEndpointName, zEndpointProtocol, zEndpointName and arcIndexId are mandatory parameters, otherwise the ObjectId is not searched for and error code NVOT_ARC_INVALID_INDEX is set.

The arcNameBinding parameter helps in identifying the arc endpoints. The arcNameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters.

If the get operation fails, the routine returns ovwNullObjectId and the error internal variable is set.

If the get operation completes successfully but the arc has not been created in the OVW database, ovwNullObjectId is returned; however the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the arc exists in the OVW database, the routine returns its OVwObjectId.

Parameters

arcNameBinding
Specifies the class of the objects in each endpoint of the arc. The endpoint can be a vertex or a graph. The allowed values are:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates either of the endpoints are vertices.

ARC_VERTEX_GRAPH_NAME_BINDING
: aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
: aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
: Indicates either of the endpoints are graphs.

Any value other than those listed is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING is returned.

aEndpointProtocol
: Specifies the protocol of the object identified as the aEndpoint of this arc. If aEndpoint is to be a vertex, aEndpointProtocol must be set with a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, aEndpoint is a graph, and aEndpointProtocol is a pointer to a valid character string in memory.

aEndpointName
: Specifies the name of the object identified as the aEndpoint of this arc. The aEndpointName together with aEndpointProtocol provide the information required to identify the object at the aEndpoint of this arc.

zEndpointProtocol
: Specifies the protocol of the zEndpoint of the arc. It works the same as the aEndpointProtocol.

zEndpointName
: Specifies the name of the zEndpoint of the arc. It works the same as the aEndpointName. Refer to the explanation above.

arcIndexId
: Specifies an integer value which distinguishes one arc from others between the same endpoints.

  It is possible to connect the same two endpoints with several arcs. This parameter provides a way to distinguish each arc named by the same endpoints.

Return Values

OVwObjectId
: OVwObjectId is an unsigned integer type. If the get operation finds the object, a positive value is returned. However, if ovwNullObjectId is returned, the error internal variable should be checked because an error might have occurred.

  An ovwNullObjectId value can be tested with the macro OVwIsIdNull as defined in the OV/ovw_types.h file.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes are set by this call:

[NVOT_SUCCESS]
: Successful operation.

[NVOT_ARC_INVALID_INDEX]
: The arc index is not valid. An arc protocol must be a positive integer and an arc name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX]
: The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]	The zEndPoint graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_INVALID_NAME_BINDING]	The name binding is not valid. It must be a number defined in the nvotTypes.h file.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example below gets the OVwObjectId of the arc created in the example given in the routine “nvotCreateArcInGraph(3)” on page 221

```
#include <nvot.h>

nvotReturnCode         rc;
OVwObjectId            arcOid = ovwNullObjectId;

nvotProtocolType       oneEndpoint.vertexProtocol = STARLAN;
char         *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType       otherEndpoint.vertexProtocol = STARLAN;
char         *         otherEndpointName = "My_Vertex_V2;
int                    arcNumber = 1;

  arcOid = nvotGetArcObjectId (ARC_VERTEX_VERTEX_NAME_BINDING,
                               oneEndpoint,
                               oneEndpointName,
                               otherEndpoint,
                               otherEndpointName,
                               arcNumber);

  if (OVwIsIdNull (arcOid))
    /* We may have had a problem. */
    if (nvotGetError () != NVOT_SUCCESS)
      printf ("Error message = %s.\n", nvotGetErrorMsg (nvotGetError()));
  else
    /* OK!...*/
    printf ("Arc %s_%s ObjectId = %d.\n", oneEndpointName, otherEndpointName,
                                                                    arcOid);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateArcInGraph(3)” on page 221.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetBoxesInGraph(3)

Purpose

Gets a list of all boxes contained in a graph

Syntax

```
#include <nvot.h>

nvotBoxList   *   nvotGetBoxesInGraph (
                      nvotGraphProtocolType   graphProtocol,
                      char          *         graphName)
```

Description

The nvotGetBoxesInGraph routine issues a get operation of all boxes contained in the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the routine does not search for a box list and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the containing graph does not exist, the routine does not search for the boxes and the error code NVOT_GRAPH_DOES_NOT_EXIST is set.

If the get operation fails, the routine returns NULL and the error variable is set. See the following error codes and return values sections.

If the get operation completes successfully but no box exists within the parent graph, the routine returns NULL and the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and boxes exist within the parent graph, the routine returns a list of all boxes contained in the identified graph.

The interface allocates structured data in memory and returns a pointer to it. The user must call one of the free memory routines to have all data de-allocated.

Parameters

graphProtocol — Specifies the protocol of the graph this routine looks at. This is the parent graph for whose boxes the routine searches. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the graph this routine looks at. Both the graphName and graphProtocol parameters are required to uniquely identify the containing graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

Return Values

nvotBoxList

Upon completion of the get operation, this routine returns a list of all boxes that are members of the identified graph.

As defined in the file nvotTypes.h, nvotBoxList is an array list of nvotBox structures. The nvotBoxList return value is a structure made up of a pointer to the first nvotBox element and an integer variable count indicating the number of elements in the list.

Each nvotBox element is a structure carrying information about a box graph. For more information, please see “Basic Structures” on page 1061.

The variable operation returned in each nvotBox element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all boxes contained in the graph created in the example in “nvotCreateBoxInGraph(3)” on page 227.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotBoxList     *      myBoxes = NULL;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_Root_Graph";

  if ((myBoxes = nvotGetBoxesInGraph (myGraphProt,
                                      myGraphName) != NULL) {

    /* OK, it seems we have gotten a few boxes. Print their names. */
    printf ("Graph %s contains %d boxes.\n", myGraphName, myBoxes->count);
    for (i = 1; i = myBoxes->count; i++)
      printf ("Box %d = %s.\n", i, myBoxes->box[i-1]..boxAttr.graphName);

    /* We don't need them any longer. Let's release all memory. */
    nvotFreeBoxList (myBoxes);
  } else {
    /* No, we've gotten no boxes. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("Graph %s contains no boxes.\n", myGraphName);
    else {
      printf ("Error occurred getting boxes from graph %s\n", myGraphName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotFree(3)” on page 304.
- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetBoxObjectId(3)

Purpose

Gets a box graph Object ID from the OVW database

Syntax

```
#include <nvot.h>

OVwObjectId   nvotGetBoxObjectId (
                    nvotGraphProtocolType   boxProtocol,
                    char          *         boxName)
```

Description

Objects in the OVW database are given object IDs. As soon as they are created in the GTM database, gtmd sends a notification to ovwdb, which then creates an ObjectId in its own database. These actions are not synchronous and the time elapsed between GTMd creation and OVwDb creation might vary, depending on activity. This is a consideration for applications that directly connect to the OVW database with the intention of adding new variables to that database.

This routine returns the OVwObjectId of the box graph identified by boxProtocol and boxName.

Protocol and Name are required parameters. If they are not specified, the ObjectId is not searched for and the error code NVOT_BOX_INVALID_INDEX is set.

If the get operation fails, the routine returns ovwNullObjectId and the error internal variable is set.

If the get operation completes successfully but the box graph has not been created in the OVW database, ovwNullObjectId is returned; however, the error code NVOT_SUCCESS is set.

If the get operation completes successfully and the box graph exists in the OVW database, the routine returns its OVwObjectId.

Parameters

boxProtocol
Specifies the protocol of the box graph. For more information on how to specify a graph protocol, refer to the file /usr/OV/conf.oid_to_protocol.

boxName
Specifies the name of the box graph. The boxName and the boxProtocol make up unique information required to identify the box graph in the GTM database.

Return Values

OVwObjectId OVwObjectId is an unsigned integer type. If the get operation finds the object, a positive value is returned. However, if ovwNullObjectId is returned, the error internal variable should be checked because an error might have occurred.

A ovwNullObjectId value can be tested with the macro OVwIsIdNull as defined in the OV/ovw_types.h file.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes are set by this call:

[NVOT_SUCCESS]	Successful operation.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box protocol must be a positive integer and a box name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_OVW_TIMED_OUT]	NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

Examples

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

The following example gets the OVwObjectId of the box graph created in the example given in the routine “nvotCreateBoxInGraph(3)” on page 227

```
#include <nvot.h>

nvotReturnCode         rc;
OVwObjectId            boxOid = ovwNullObjectId;

nvotGraphProtocolType  boxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         boxName = "My_Box_STARLAN_Graph";

  boxOid = nvotGetBoxObjectId (boxProt, boxName);

  if (OVwIsIdNull (boxOid))
    /* We may have had a problem. */
    if (nvotGetError () != NVOT_SUCCESS)
      printf ("Error message = %s.\n", nvotGetErrorMsg (nvotGetError()));
  else
    /* OK!...*/
    printf ("Box graph %s ObjectId = %d.\n", boxName, boxOid);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetBoxesWhichVertexIsMemberOf(3)

Purpose

Gets a list of all boxes of which a vertex is member

Syntax

```
#include <nvot.h>

nvotBoxList * nvotGetBoxesWhichVertexIsMemberOf (
                    nvotVertexProtocolType   vertexProtocol,
                    char          *          vertexName)
```

Description

A vertex symbol can be displayed in the submap of several box graphs. In other words, a vertex object can be a member of several box graph objects in the GTM database.

This routine returns a list of all boxes of which the vertex identified by vertexProtocol and vertexName is a member.

The protocol and name parameters uniquely identify objects in the GTM database. Therefore vertexProtocol and vertexName parameters are mandatory. If these parameters are not specified, a box list is not searched for and the error code NVOT_VERTEX_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error internal variable is set.

If the get operation completes successfully but the vertex is not a member of any box, NULL is also returned; however the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the vertex is a member of any box, the routine returns a list of all boxes with which the vertex is associated.

Notice that the interface allocates structured data in memory and returns a pointer to it. The user must call the nvotFreeBoxList routine in order to have all data deallocated.

Parameters

vertexProtocol
Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName
Specifies the name of the vertex. The name can be any string of characters. However, once specified, the same name must be used in any reference to this vertex.

Return Values

nvotBoxList Upon completion of the get operation, routine nvotGetBoxesWhichVertexIsMemberOf returns a list of all boxes that are members of the identified graph.

As defined in the file nvotTypes.h, *nvotBoxList* is an array list of *nvotBox* structures. The *nvotBoxList* value is a structure made up of a pointer to the first *nvotBox* element and an integer variable *count* indicating the number of elements in the list.

Each *nvotBox* element is a structure carrying the actual information about a box graph. See the nvotTypes.h file for additional information.

Note: The variable *operation* returned in each *nvotBox* element has no meaning in this routine.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes are set by this call:

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example issues an operation to gtmd to get all parent boxes of the vertex created in the example given in the routine “nvotCreateBoxInGraph(3)” on page 227 and prints each graphName string.

```
#include <nvot.h>

int                     i;
nvotReturnCode          rc;
nvotBoxList      *      parentBoxes = NULL;
nvotBox          *      boxObj;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";

  if ((parentBoxes = nvotGetBoxesWhichVertexIsMemberOf (
                                                  myVertexProt,
                                                  myVertexName) != NULL) {

    /* OK, it seems we have gotten a few boxes. Print their names. */
    printf ("Vertex %s is member of %d boxes.\n", myVertexName,
                                                       parentBoxes.count);
    for (i = 1; i < parentBoxes->count; i++) {
      boxObj = &parentBoxes->graph[i]; /* Get pointer to the next box. */
      printf ("Graph %d = %s.\n", i, boxObj->graphAttr.graphName);
```

```
  }
}
else {
  /* No, we've gotten no graphs. What happened?... */
  if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
    printf ("Vertex %s is not member of any box.\n", myVertexName);
  else {
    printf ("Error occurred getting parent boxes of vertex %s.\n",
                                                          myVertexName);
    printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
  }
}
/* We don't need them any longer. Let's release all memory. */
nvotFreeBoxList (parentBoxes);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetError(3)

Purpose

Returns the error code set by the last function call

Syntax

```
#include <nvot.h>
nvotReturnCode  nvotGetError ()
```

Description

The interface has an internal error variable that is set for every function call. The variable is reset to [NVOT_SUCCESS] before a function call and set to the error code that the function call returns when the call is processed.

The nvotGetError routine returns the error code set at the last function call.

Return Values

If the last function call was successful, nvotGetError returns [NVOT_SUCCESS]. If the last function call was unsuccessful, nvotGetError returns the error code that the call returned.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error variable is reset upon entering and set before exiting this call to the API. All possible error codes set by the nvot* calls and their related message strings are:

[NVOT_SUCCESS] Successful operation.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_INVALID_VERTEX_PROTOCOL]
Invalid vertex protocol. The vertex protocol must be a positive integer.

[NVOT_INVALID_LAYOUT] Invalid layout. The layout must be a number defined in the nvotTypes.h file.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_INVALID_VERTEX_TYPE] The vertex type is not valid. A vertex type must be a number defined in the nvotTypes.h file.

[NVOT_INVALID_ARC_TYPE] The arc type is not valid. An arc type must be a number defined in the nvotTypes.h file.

[NVOT_A_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the A endpoint of the arc does not exist in the GTM database.

[NVOT_Z_ENDPOINT_GRAPH_DOES_NOT_EXIST]
The graph defined as the Z endpoint of the arc does not exist in the GTM database.

[NVOT_INVALID_NAME_BINDING] The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ARC_INVALID_INDEX] The arc index is not valid. It must be a positive integer.

[NVOT_ROOT_GRAPH_DOES_NOT_EXIST]
The root graph does not exist. A root graph must be created before issuing this call.

[NVOT_GRAPH_ALREADY_EXIST] A graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_BOX_ALREADY_EXIST] A box already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_ROOT_GRAPH_ALREADY_EXIST]
A root graph already exists with the same protocol and name for which this call is attempting to create a graph, box, or root graph.

[NVOT_OTHER_TYPE_GRAPH_EXIST]
Another type of graph exists. This call is attempting to create a graph, box, or root graph with a protocol and name already used for a graph of type INVALID or OTHER.

[NVOT_GTMD_CONNECTION_ERROR]
There is a GTM connection error. The connection cannot be opened. Issue the nvotInit routine again.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

[NVOT_OVW_CONNECTION_ERROR]
There is an connection error with the NetView for AIX program. The connection to the object database cannot be opened.

[NVOT_OVW_TIMED_OUT] NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol and/or name must not be NULL.

[NVOT_ALREADY_INITIALIZED] Already initialized. The routine attempted to re-establish the connection with tmdd but the connection is still open.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_VERTEX_IS_NOT_MEMBER] The routine is attempting to create or change an arc but the endpoint vertex is not a member of the parent graph.

[NVOT_GRAPH_IS_NOT_MEMBER] The routine is attempting to create or change an arc but the endpoint graph is not a member of the parent graph.

[NVOT_GRAPHS_LAYOUT_IS_NOT_STAR]
The routine is attempting to center a child graph or box in a layout that is not STAR_LAYOUT.

[NVOT_PROTOCOL_WAS_NOT_REGISTERED]
The protocol was not registered in the /usr/OV/conf/oid_to_protocol file.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotGetErrorMsg(3)” on page 326.

nvotGetErrorMsg(3)

Purpose

Returns a message string related to a nvotReturnCode

Syntax

```
#include <nvot.h>
char   *   nvotGetErrorMsg (nvotReturnCode returnCode)
```

Description

This routine returns a message string related to the nvotReturnCode passed to the returnCode parameter.

Parameters

returnCode
A nvotReturnCode type variable returned by a function call or by the nvotGetError routine.

Return Values

*char **	The message string associated with the returnCode value passed.

Error Codes

None.

For a complete list of error codes and their associated messages, see the error codes section in “nvotGetError(3)” on page 323.

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotGetError(3)” on page 323.

nvotGetGraphObjectId(3)

Purpose

Gets a graph Object ID from the OVW database

Syntax

```
#include <nvot.h>

OVwObjectId   nvotGetGraphObjectId (
                    nvotGraphProtocolType   graphProtocol,
                    char          *         graphName)
```

Description

Objects in the OVW database are given object IDs. As soon as objects are created in the GTM database, gtmd sends a notification to ovwdb, which creates an ObjectId in its own database. These actions are not synchronous, and the elapsed time between gtmd creation and OVwDb creation might vary, depending on activity. This is a consideration for applications that directly connect to the OVW database with the intention of adding new variables to that database.

This routine returns the OVwObjectId of the graph identified by graphProtocol and graphName.

Protocol and name are mandatory parameters. If these parameters are not specified, the ObjectId is not searched for and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the get operation fails, the routine returns ovwNullObjectId and the error internal variable is set.

If the get operation completes successfully but the graph has not been created in OVwDb, ovwNullObjectId is returned; however the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the graph exists in the OVW database, the routine returns its OVwObjectId.

Parameters

graphProtocol
Specifies the protocol of the graph. For more information on how to specify a graph protocol refer to the file /usr/OV/conf.oid_to_protocol.

graphName
Specifies the name of the graph. The graphName, together with graphProtocol, makes up unique information required to identify the graph in the GTM database.

Return Values

OVwObjectId
OVwObjectId is an unsigned integer type. If the get operation finds the object, a positive value is returned. If ovwNullObjectId is returned, the error internal variable should be checked because an error might have occurred. Refer to the Description of this API for more information.

An ovwNullObjectId value can be tested with the macro OVwIsIdNull as defined in OV/ovw_types.h file.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes are set by this call:

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_OVW_TIMED_OUT]	NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Try increasing the timeout value.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example gets the OVwObjectId of the graph created in the example given in the routine “nvotCreateGraphInGraph(3)” on page 235

```
#include <nvot.h>

nvotReturnCode         rc;
OVwObjectId            graphOid = ovwNullObjectId;

nvotGraphProtocolType  graphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         graphName = "My_Child_SDLC_Graph";

  graphOid = nvotGetGraphObjectId (graphProt, graphName);

  if (OVwIsIdNull (graphOid))
    /* We may have had a problem. */
    if (nvotGetError () != NVOT_SUCCESS)
      printf ("Error message = %s.\n", nvotGetErrorMsg (nvotGetError()));
  else
    /* OK!...*/
    printf ("Graph %s ObjectId = %d.\n", graphName, graphOid);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetGraphsInGraph(3)

Purpose

Gets a list of all graphs contained in a parent graph

Syntax

```
#include <nvot.h>

nvotGraphList  *  nvotGetGraphsInGraph (
                      nvotGraphProtocolType   graphProtocol,
                      char          *         graphName)
```

Description

The nvotGetGraphsInGraph routine issues a get operation of all graphs contained in the parent graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, the routine does not search for a graph list and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the containing graph does not exist, the routine does not search for the graphs and the error code NVOT_GRAPH_DOES_NOT_EXIST is set.

If the get operation fails, the routine returns NULL and the error variable is set. See the following error codes and return values sections.

If the get operation completes successfully but no graph exists within the parent graph, the routine returns NULL and the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and graphs exist within the parent graph, the routine returns a list of all graphs contained in the identified graph.

The interface allocates structured data in memory and returns a pointer to it. The user must call one of the memory free routines to have all data de-allocated.

Parameters

graphProtocol — Specifies the protocol of the graph this routine looks at. This is the parent graph for whose child graphs this routine searches. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName — Specifies the name of the graph this routine looks at. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

Return Values

nvotGraphList — Upon completion of the get operation, the nvotGetGraphsInGraph routine returns a list of all graphs members of the identified graph.

As defined in the file nvotTypes.h, nvotGraphList is an array list of nvotGraph

structures. The nvotGraphList return value is a structure made up of a pointer to the first nvotGraph element and an integer variable count indicating the number of elements in the list.

Each nvotGraph element is a structure carrying the actual information about a graph. For more information, refer to the information about basic structures in the *NetView for AIX Programmer's Guide* and the file nvotTypes.h.

The variable operation returned in each nvotGraph element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol and/or name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	A graph does not exist in the GTM database.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all graphs contained in the graph created in the example in “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotGraphList    *     myGraphs = NULL;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.17";
char         *         myGraphName = "My_Root_Graph";

  if ((myGraphs = nvotGetGraphsInGraph (myGraphProt,
                                        myGraphName) != NULL) {

    /* OK, it seems we have gotten a few graphs. Print their names. */
    printf ("Graph %s contains %d arcs.\n", myGraphName, myGraphs->count);
    for (i = 1; i = myGraphs->count; i++)
      printf ("Graph %d = %s.\n", i, myGraphs->graph[i-1].graphAttr.graphName);

    /* We don't need them any longer. Let's release all memory. */
    nvotFreeGraphList (myGraphs);
```

```
} else {
  /* No, we've gotten no graphs. What happened?... */
  if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
    printf ("Graph %s contains no graphs.\n", myGraphName);
  else {
    printf ("Error occurred getting graphs from graph %s\n", myGraphName);
    printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
  }
}
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotFree(3)” on page 304.
- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetGraphsWhichArcIsMemberOf(3)

Purpose

Gets a list of all graphs of which an arc is a member

Syntax

```
#include <nvot.h>

nvotGraphList * nvotGetGraphsWhichArcIsMemberOf (
                     nvotNameBindingType     nameBinding,
                     nvotProtocolType        aEndpointProtocol,
                     char          *         aEndpointName,
                     nvotProtocolType        zEndpointProtocol,
                     char          *         zEndpointName,
                     int                     arcIndexId)
```

Description

An arc symbol can be displayed in the submap of several graphs. In other words, an arc object can be a member of several graph objects in the GTM database.

This routine returns a list of all graphs of which the arc identified by aEndpoint, zEndpoint, and arcIndexId is a member.

An arc might exist to connect two vertices, two graphs, or a vertex to a graph and vice versa. These are called arc endpoints. An arc is recognized and referenced by its aEndpoint, zEndpoint, and arcIndexId.

The nameBinding parameter helps in identifying the arc endpoints. The nameBinding must always be compatible with the values passed in the aEndpointProtocol and zEndpointProtocol parameters. Only ARC_VERTEX_VERTEX_NAME_BINDING, ARC_VERTEX_GRAPH_NAME_BINDING, ARC_GRAPH_VERTEX_NAME_BINDING, and ARC_GRAPH_GRAPH_NAME_BINDING are accepted for nameBinding. Should any other value be passed, the routine returns NULL and the error internal variable is set to NVOT_INVALID_NAME_BINDING.

The nvotProtocolType is a combination of an enumerated type and a char pointer as defined in nvotTypes.h file. Special care must be taken when specifying aEndpointProtocol and zEndpointProtocol; assigning these variables a nvotVertexProtocolType value when arcNameBinding identifies the endpoint as a graph causes an error. This error is equivalent to asking the GTM interface to take an integer value for a char pointer and leads to unpredictable errors.

All parameters in this routine are mandatory. If they are not specified, a graph list is not searched for and the error code NVOT_ARC_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error internal variable is set.

If the get operation completes successfully but the arc is not a member of any graph, NULL is also returned; however, the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the arc is a member of any graph, the routine returns a list of all graphs with which the arc is associated.

Notice that the interface allocates structured data in memory and returns a pointer to it. The user must call the nvotFreeGraphList routine in order to have all data de-allocated.

Parameters

nameBinding
Specifies the class of the objects in each endpoint of the arc. The endpoint can be a vertex or a graph. The allowed values are:

ARC_VERTEX_VERTEX_NAME_BINDING
Indicates either of the endpoints are vertices.

ARC_VERTEX_GRAPH_NAME_BINDING
aEndpoint is a vertex and zEndpoint is a graph.

ARC_GRAPH_VERTEX_NAME_BINDING
aEndpoint is a graph and zEndpoint is a vertex.

ARC_GRAPH_GRAPH_NAME_BINDING
Indicates either of the endpoints are graphs.

Any value other than those listed is rejected by the interface and the error code NVOT_INVALID_NAME_BINDING is set.

Regardless of the choice made in the initialization session, nameBinding is an identification of what value is set in the aEndpointProtocol and zEndpointProtocol variables.

aEndpointProtocol
Specifies the protocol of the object identified as the aEndpoint of the arc. If aEndpoint is to be a vertex, aEndpointProtocol must be set with a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, aEndpoint is a graph, and aEndpointProtocol is a pointer to a valid character string in memory.

aEndpointName
Specifies the name of the object identified as the aEndpoint of the arc. The aEndpointName, together with aEndpointProtocol, is the information required to identify the object at the aEndpoint of this arc. It can be any string of characters. However, once specified, the same name must be used in any reference to this graph.

zEndpointProtocol
Specifies the protocol of the zEndpoint of the arc. If zEndpoint is to be a vertex, zEndpointProtocol must be set with a value from the enumerated type nvotVertexProtocolType defined in the file nvotTypes.h. Otherwise, zEndpoint is a graph, and zEndpointProtocol is a pointer to a valid character string in memory.

zEndpointName
Specifies the name of the zEndpoint of the arc. The zEndpointName, together with zEndpointProtocol, is the information required to identify the object at the zEndpoint of the arc. It can be any string of characters. However, once specified, the same name must be used in any reference to this graph.

arcIndexId
Specifies an integer value which distinguishes one arc from others between the same endpoints. It is possible to connect the same two endpoints with several arcs; this parameter provides a way to distinguish each arc named by the same endpoints.

Return Values

nvotGraphList Upon completion of the get operation, nvotGetGraphsWhichArcIsMemberOf returns a list of all graphs of which the arc is a member.

As defined in the file nvotTypes.h, **nvotGraphList** is an array list of ***nvotGraph*** structures. **nvotGraphList** is a structure made up of a pointer to the first ***nvotGraph*** element and an integer variable ***count*** indicating the number of elements in the list.

Each ***nvotGraph*** element is a structure carrying the actual information about a graph.

Note: The variable ***operation*** returned in each ***nvotGraph*** element has no meaning in this routine.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS]	Successful operation.
[NVOT_ARC_INVALID_INDEX]	The arc index is not valid. An arc protocol must be a positive integer and an arc name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_GTMD_INVALID_RESPONSE]	GTM invalid response. A query to a graph or member table returned an unexpected response from GTMd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example below gets all the parent graphs of the arc created in the example given in the routine “nvotCreateArcInGraph(3)” on page 221.

```
#include <nvot.h>

int                     i;
nvotReturnCode          rc;
nvotGraphList           parentGraphs = NULL;
nvotGraph         *     graphObj;

nvotProtocolType        oneEndpoint.vertexProtocol = STARLAN;
char          *         oneEndpointName = "My_Vertex_V1";
nvotProtocolType        otherEndpoint.vertexProtocol = STARLAN;
char          *         otherEndpointName = "My_Vertex_V2;
int                     arcNumber = 1;

  if ((parentGraphs = nvotGetGraphsWhichArcIsMemberOf (
                                              ARC_VERTEX_VERTEX_NAME_BINDING,
                                              oneEndpoint,
                                              oneEndpointName,
                                              otherEndpoint,
                                              otherEndpointName,
                                              arcNumber);

    /* OK, it seems we have gotten a few graphs. Print their names. */
    printf ("Arc %s_%s is member of %d graphs.\n", oneEndpointName,
                                                   otherEndpointName,
                                                   parentGraphs.count);
    for (i = 1; i < parentGraphs->count; i++) {
      graphObj = &parentGraphs->graph[i]; /* Get pointer to the next graph. */
      printf ("Graph %d = %s.\n", i, graphObj->graphAttr.graphName);
    }
  }
  else {
    /* No, we've gotten no graphs. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("Arc %s_%s is not member of any graph.\n", oneEndpointName,
                                                         otherEndpointName);
    else {
      printf ("Error occurred getting parent graphs of arc %s_%s.\n",
                                                            oneEndpointName,
                                                            otherEndpointName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
  /* We don't need'em any longer. Let's release all memory. */
  nvotFreeGraphList (parentGraphs);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotSetSynchronousCreation(3)” on page 368.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetGraphsWhichBoxIsMemberOf(3)

Purpose

Gets a list of all graphs of which a box is a member

Syntax

```
#include <nvot.h>

nvotGraphList * nvotGetGraphsWhichBoxIsMemberOf (
                     nvotGraphProtocolType   boxProtocol,
                     char          *         boxName)
```

Description

A box symbol can be displayed in the submap of several other graphs. In other words, a box graph object can be a member of several other graph objects in the GTM database.

This routine returns a list of all graphs of which the box identified by boxProtocol and boxName is a member.

The protocol and name parameters uniquely identify objects in the GTM database. Therefore, the boxProtocol and boxName parameters are mandatory. If these parameters are not specified, a graph list is not searched for and the error code NVOT_BOX_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error internal variable is set.

If the get operation completes successfully but the box is not a member of any other graph, NULL is also returned; however, the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the box is a member of other graphs, the routine returns a list of all graphs with which the box is associated.

Notice that the interface allocates structured data in memory and returns a pointer to it. The user must call the nvotFreeGraphList routine in order to have all data deallocated.

Parameters

boxProtocol
: Specifies the protocol of the box this routine searches. In other words, the box whose parent graphs are searched for. For more information refer to the file /usr/OV/conf/oid_to_protocol.

boxName
: Specifies the name of the box this routine searches. The boxName, together with boxProtocol, provides the information required to identify the box graph. The boxName can be any string of characters; however, once specified, the same name must be used in any reference to this box graph.

Return Values

nvotGraphList Upon completion of the get operation, this routine returns a list of all graphs of which the box is a member.

As defined in the file nvotTypes.h, **nvotGraphList** isan array list of ***nvotGraph*** structures. The **nvotGraphList** is a structure made up of a pointer to the first ***nvotGraph*** element and an integer variable ***count*** indicating the number of elements in the list.

Each ***nvotGraph*** element is a structure carrying the actual information about a graph.

Note: The variable ***operation*** returned in each ***nvotGraph*** element has no meaning in this routine.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS]	Successful operation.
[NVOT_BOX_INVALID_INDEX]	The box index is not valid. A box protocol must be a positive integer and a box name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example gets from GTMd all parent graphs of the box created in the example given in the routine “nvotCreateBoxInGraph(3)” on page 227.

```
#include <nvot.h>

int                    i;
nvotReturnCode         rc;
nvotGraphList          parentGraphs = NULL;
nvotGraph        *     graphObj;

nvotGraphProtocolType  STARLAN_BoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         STARLAN_BoxName = "My_Box_STARLAN_Graph";

  if ((parentGraphs = nvotGetGraphsWhichBoxIsMemberOf (
                                                  STARLAN_BoxProt
                                                  STARLAN_BoxName) != NULL) {

    /* OK, it seems we have gotten a few graphs. Print their names. */
    printf ("Box %s is member of %d graphs.\n", STARLAN_BoxName,
                                                      parentGraphs.count);
    for (i = 1; i < parentGraphs->count; i++) {
      graphObj = &parentGraphs->graph[i]; /* Get pointer to the next graph. */
      printf ("Graph %d = %s.\n", i, graphObj->graphAttr.graphName);
    }
  }
  else {
    /* No, we've gotten no graphs. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("Box %s is not member of any graph.\n", STARLAN_BoxName);
    else {
      printf ("Error occurred getting parent graphs of box %s.\n",
                                                        STARLAN_BoxName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
  /* We don't need'em any longer. Let's release all memory. */
  nvotFreeGraphList (parentGraphs);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetGraphsWhichGraphIsMemberOf(3)

Purpose

Gets a list of all graphs of which a child graph is a member

Syntax

```
#include <nvot.h>

nvotGraphList * nvotGetGraphsWhichGraphIsMemberOf (
                     nvotGraphProtocolType   graphChildProtocol,
                     char          *         graphChildName)
```

Description

A graph symbol can be displayed in the submap of several other graphs. In other words, a graph object can be a member of several other graph objects in the GTM database.

This routine returns a list of all graphs of which the child graph identified by graphChildProtocol and graphChildName is a member.

The protocol and name parameters uniquely identify objects in the GTM database. Therefore, the graphChildProtocol and graphChildName parameters are mandatory. If the parmeters are not specified, a graph list is not searched for and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error internal variable is set.

If the get operation completes successfully but the child graph is not a member of any other graph, NULL is also returned; however, the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the child graph is a member of other graphs, the routine returns a list of all graphs with which the child graph is associated.

Note that the interface allocates structured data in memory and returns a pointer to it. The user must call the nvotFreeGraphList routine in order to have all data deallocated.

Parameters

graphChildProtocol
: Specifies the protocol of the child graph this routine searches (in other words, the graph whose parent graphs are searched for). For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphChildName
: Specifies the name of the child graph this routine searches. The graphChildName, together with graphChildProtocol, provides the information required to identify the child graph. It name can be any string of characters. However, once specified, the same name must be used in any reference to this graph.

Return Values

nvotGraphList

Upon completion of the get operation, nvotGetGraphsWhichGraphIsMemberOf returns a list of all graphs of which the child graph is a member.

As defined in the file nvotTypes.h, **nvotGraphList** is an array list of ***nvotGraph*** structures. The **nvotGraphList** is a structure made up of a pointer to the first ***nvotGraph*** element and an integer variable ***count*** indicating the number of elements in the list.

Each ***nvotGraph*** element is a structure carrying the actual information about a graph. See the nvotTypes.h file for additional information.

Note: The variable ***operation*** returned in each ***nvotGraph*** element has no meaning in this routine.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. All possible error codes set by this call and their related message strings are:

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol must be a positive integer and a graph name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example below fetches from GTMd all parent graphs of the graph created in the example given in the routine “nvotCreateGraphInGraph(3)” on page 235.

```
#include <nvot.h>

int                     i;
nvotReturnCode          rc;
nvotGraphList           parentGraphs = NULL;
nvotGraph         *     graphObj;

nvotGraphProtocolType   mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";
char          *         myChildSDLCGraphName = "My_Child_SDLC_Graph";

  if ((parentGraphs = nvotGetGraphsWhichGraphIsMemberOf (
                                            mySDLCGraphsProt,
                                            myChildSDLCGraphName) != NULL) {

    /* OK, it seems we have gotten a few graphs. Print their names. */
    printf ("Graph %s is member of %d graphs.\n", myChildSDLCGraphName,
                                                    parentGraphs.count);
    for (i = 1; i < parentGraphs->count; i++) {
      graphObj = &parentGraphs->graph[i]; /* Get pointer to the next graph. */
      printf ("Graph %d = %s.\n", i, graphObj->graphAttr.graphName);
    }
  }
  else {
    /* No, we've gotten no graphs. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("myChildSDLCGraphName %s is not member of any graph.\n",
                                                      myChildSDLCGraphName);
    else {
      printf ("Error occurred getting parent graphs of graph %s.\n",
                                                      myChildSDLCGraphName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
  /* We don't need'em any longer. Let's release all memory. */
  nvotFreeGraphList (parentGraphs);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetGraphsWhichVertexIsMemberOf(3)

Purpose

Gets a list of all graphs of which a vertex is member

Syntax

```
#include <nvot.h>

nvotGraphList * nvotGetGraphsWhichVertexIsMemberOf (
                    nvotVertexProtocolType   vertexProtocol,
                    char          *          vertexName)
```

Description

A vertex symbol can be displayed in the submap of several graphs. In other words, a vertex object can be a member of several graph objects in the GTM database.

This routine returns a list of all graphs of which the vertex identified by vertexProtocol and vertexName is a member.

The protocol and name parameters uniquely identify objects in the GTM database. Therefore, the vertexProtocol and vertexName parameters are mandatory. If these parameters are not specified, a graph list is not searched for and the error code NVOT_VERTEX_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error internal variable is set.

If the get operation completes successfully but the vertex is not a member of any graph, NULL is also returned; however, the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the vertex is a member of any graph, the routine returns a list of all graphs with which the vertex is associated.

Note that the interface allocates structured data in memory and returns a pointer to it. The user must call the nvotFreeGraphList routine in order to have all data deallocated.

Parameters

vertexProtocol
Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName
Specifies the name of the vertex. It can be any string of characters. However, once specified, the same name must be used in any reference to this vertex.

Return Values

nvotGraphList Upon completion of the get operation, the nvotGetGraphsWhichVertexIsMemberOf routine returns a list of all graphs of which the vertex is a member.

As defined in the file nvotTypes.h, **nvotGraphList** is an array list of ***nvotGraph*** structures. The **nvotGraphList** is a structure made up of a pointer to the first ***nvotGraph*** element and an integer variable ***count*** indicating the number of elements in the list.

Each ***nvotGraph*** element is a structure carrying the actual information about a graph. See the nvotTypes.h file for more information.

Note: The ***operation*** returned in each ***nvotGraph*** element has no meaning in this routine.

Error Codes

Upon return, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes are set by this call:

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The example below issues an operation to GTMd to GET all parent graphs of the vertex created in the example given in the routine “nvotCreateVertexInGraph(3)” on page 265 and prints out each graphName string.

```
#include <nvot.h>

int                     i;
nvotReturnCode          rc;
nvotGraphList    *      parentGraphs = NULL;
nvotGraph        *      graphObj;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";

  if ((parentGraphs = nvotGetGraphsWhichVertexIsMemberOf (
                                                    myVertexProt,
                                                    myVertexName) != NULL) {

    /* OK, it seems we have gotten a few graphs. Print their names. */
    printf ("Vertex %s is member of %d graphs.\n", myVertexName,
                                                        parentGraphs.count);
    for (i = 1; i < parentGraphs->count; i++) {
      graphObj = &parentGraphs->graph[i]; /* Get pointer to the next graph. */
```

```
    printf ("Graph %d = %s.\n", i, graphObj->graphAttr.graphName);
  }
}
else {
  /* No, we've gotten no graphs. What happened?... */
  if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
    printf ("Vertex %s is not member of any graph.\n", myVertexName);
  else {
    printf ("Error occurred getting parent graphs of vertex %s.\n",
                                                          myVertexName);
    printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
  }
}
/* We don't need'em any longer. Let's release all memory. */
nvotFreeGraphList (parentGraphs);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326. :

nvotGetSapsOnVertex(3)

Purpose

Gets a list of all SAPs associated with a vertex

Syntax

```
#include <nvot.h>

nvotSapList   *   nvotGetSapsOnVertex (
                      nvotVertexProtocolType   vertexProtocol,
                      char          *          vertexName)
```

Description

The nvotGetSapsOnVertex routine issues a get operation of all SAPs, used and provided, by the vertex identified by vertexProtocol and vertexName.

The protocol and name parameters uniquely identify objects in the GTM database. The vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the routine does not search for a SAP list and the error code NVOT_VERTEX_INVALID_INDEX is set.

If the get operation fails, the routine returns NULL and the error variable is set. See the following error codes and return values sections.

If the get operation completes successfully but no SAP is associated with the vertex, the routine returns NULL and the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and SAPs are found, the routine returns a list of all SAPs associated with the identified vertex.

The interface allocates structured data in memory and returns a pointer to it. The user must call one of the memory free routines to have all data de-allocated.

Parameters

vertexProtocol — Specifies the protocol of the vertex this routine looks at. This is the vertex for whose SAPs the routine searches. The vertexProtocol parameter is defined in the file nvotTypes.h.

vertexName — Specifies the name of the vertex this routine looks at. Both the vertexName and vertexProtocol are required to identify the vertex using and providing SAPs. This parameter is a string of characters originally used to create the vertex and the SAPs associated with it.

Return Values

nvotSapList — Upon completion of the get operation, this routine returns a list of all SAPs used and provided by the identified vertex.

As defined in the file nvotTypes.h, nvotSapList is an array list of nvotSap structures. The nvotSapList return value is a structure made up of a pointer to the first nvotSap element and an integer variable count indicating the number of elements in the list.

Each nvotSap element is a structure carrying information about a SAP. For more information, please see “Basic Structures” on page 1061.

The variable operation returned in each nvotSap element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all SAPs used and provided by the vertex created in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>


nvotReturnCode         rc;
nvotSapList     *      mySaps = NULL;
int                    using = 0;
int                    providing = 0;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";
  if ((mySaps = nvotGetSapsOnVertex (myVertexProt,
                                     myVertexName) != NULL) {

    /* OK, it seems we have gotten a few saps. Count using and providing.*/
    for (i = 1; i = mySaps->count; i++) {
      if (mySaps->sap-&gt[i-1]..sapAttr.sapServiceType ==  USING)
        using++;
      else
        providing++;
    /* Print their names. */
    printf ("Vertex %s uses %d and provides %d saps.\n", myVertexName,
                                                          using, providing);
    for (i = 1; i = mySaps->count; i++)
      printf ("Sap %d = %s.\n", i, mySaps->sap[i-1].sapAttr.sapAddress);
```

```
  /* We don't need them any longer. Let's release all memory. */
  nvotFreeSapList (mySaps);
} else {
  /* No, we've gotten no saps. What happened?... */
  if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
    printf ("Vertex %s uses/provides no saps.\n", myVertexName);
  else {
    printf ("Error occurred getting saps from vertex %s\n", myVertexName);
    printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
  }
}
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetVertexObjectId(3)

Purpose

Gets a vertex ObjectId from the OVW database

Syntax

```
#include <nvot.h>

OVwObjectId   nvotGetVertexObjectId (
                    nvotVertexProtocolType   vertexProtocol,
                    char          *          vertexName)
```

Description

Objects in the OVW database are given objectIds. As soon as the objects are created in the GTM database, gtmd sends a notification to ovwdb, which creates an ObjectId in its own database. These actions are not synchronous. The elapsed time between GTMd creation and OVwDb creation can vary, depending on activity. This is a consideration for applications that directly connect to the OVW database with the intention of adding new variables to that database.

This routine returns the OVwObjectId of the vertex identified by vertexProtocol and vertexName.

Protocol and name are mandatory parameters. If the parameters are not specified, the ObjectId is not searched for and the error code NVOT_VERTEX_INVALID_INDEX is set.

If the get operation fails, the routine returns ovwNullObjectId and the error internal variable is set.

If the get operation completes successfully but the vertex has not been created in the OVW database, ovwNullObjectId is returned; however, the error internal variable is set to NVOT_SUCCESS.

If the get operation completes successfully and the vertex exists in the OVW database, the routine returns its OVwObjectId.

Parameters

vertexProtocol
: Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

vertexName
: Specifies the name of the vertex. It can be any string of characters. However, once specified, the same name must be used in any reference to this vertex.

Return Values

OVwObjectId
: OVwObjectId is an unsigned integer type. If the get operation finds the object, a positive value is returned. If ovwNullObjectId is returned, check the error internal variable because an error might have occurred.

  A ovwNullObjectId value can be tested with the macro OVwIsIdNull as defined in the OV/ovw_types.h file.

Error Codes

When the routine completes and returns control to its caller, an error internal variable is set. A call to the routine nvotGetError returns the error code set at the last API call. The error internal variable is reset upon entering and set before exiting this call to the API. The following error codes set by this call:

[NVOT_SUCCESS]	Successful operation.
[NVOT_VERTEX_INVALID_INDEX]	The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_OVW_TIMED_OUT]	NetView for AIX timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView for AIX database might be down. Increase the timeout value.

A printable message string is accessible through a call to the routine nvotGetErrorMsg as in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example gets the OVwObjectId of the vertex created in the example given in the routine “nvotCreateVertexInGraph(3)” on page 265

```
#include <nvot.h>

nvotReturnCode         rc;
OVwObjectId            vertexOid = ovwNullObjectId;

nvotVertexProtocolType myVertexProt = STARLAN;
char         *         myVertexName = "My_Vertex";

  vertexOid = nvotGetVertexObjectId (myVertexProt, myVertexName);

  if (OVwIsIdNull (vertexOid))
    /* We may have had a problem. */
    if (nvotGetError () != NVOT_SUCCESS)
      printf ("Error message = %s.\n", nvotGetErrorMsg (nvotGetError()));
  else
    /* OK!...*/
    printf ("Vertex %s ObjectId = %d.\n", myVertexName, vertexOid);
```

Libraries

- libnvot.a

Files

- nvot.h

Related Information

- See “nvotCreateVertexInGraph(3)” on page 265.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetVerticesInBox(3)

Purpose

Gets a list of all vertices contained in a box graph

Syntax

```
#include <nvot.h>

nvotVertexList * nvotGetVerticesInBox (
                     nvotGraphProtocolType   boxProtocol,
                     char          *         boxName)
```

Description

The nvotGetVerticesInBox routine issues a get operation of all vertices contained in the box graph identified by boxProtocol and boxName.

The protocol and name parameters uniquely identify objects in the GTM database. The boxProtocol and boxName parameters are required. If one of these parameters is not provided, a vertex list is not searched for and the error code NVOT_BOX_INVALID_INDEX is set.

If the containing box graph does not exist, the nvotGetVerticesInBox routine does not search for the vertices and the error code NVOT_BOX_DOES_NOT_EXIST is set.

If the get operation fails, the routine returns NULL and the error variable is set. See the following error codes and return values sections.

If the get operation completes successfully but no vertex exists within the box graph, the routine returns NULL but the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and vertices exist in the box, the routine returns a list of all vertices contained in the identified box.

The interface allocates structured data in memory and returns a pointer to it. The user must call one of the memory free routines to have all data de-allocated.

Parameters

boxProtocol Specifies the protocol of the box graph this routine looks at. This is the box for whose vertices the routine searches. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

boxName Specifies the name of the box graph this routine looks at. Both the boxName and boxProtocol parameters are required to uniquely identify the containing box graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this box.

Return Values

nvotVertexList

Upon completion of the get operation, the nvotGetVerticesInBox routine returns a list of all vertices members of the identified box graph.

As defined in the file nvotTypes.h, the nvotVertexList return value is an array list of nvotVertex structures. The nvotVertexList return value is a structure made up of a pointer to the first nvotVertex element and an integer variable count indicating the number of elements in the list.

Each nvotVertex element is a structure carrying information about a vertex. For more information, please see “Basic Structures” on page 1061.

The variable operation returned in each nvotVertex element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all vertices contained in the box created in the example in “nvotCreateVertexInBox(3)” on page 261.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotVertexList    *    myVertices = NULL;

nvotGraphProtocolType  myBoxProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myBoxName = "My_Box_Graph";

  if ((myVertices = nvotGetVerticesInBox (myBoxProt,
                                          myBoxName) != NULL) {

    /* OK, it seems we have gotten a few vertices. Print their names. */
    printf ("Box %s contains %d vertices.\n", myBoxName, myVertices->count);
    for (i = 1; i = myVertices->count; i++)
```

```
    printf ("Vertex %d=%s.\n", i, myVertices->vertex[i-1].vertexAttr.vertexName);

  /* We don't need them any longer. Let's release all memory. */
  nvotFreeVertexList (myVertices);

} else {
  /* No, we've gotten no vertices. What happened?... */
  if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
    printf ("Box %s contains no vertices.\n", myBoxName);
  else {
    printf ("Error occurred getting vertices from box %s\n", myBoxName);
    printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
  }
}
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotGetVerticesInGraph(3)

Purpose

Gets a list of all vertices contained in a graph

Syntax

```
#include <nvot.h>

nvotVertexList * nvotGetVerticesInGraph (
                     nvotGraphProtocolType   graphProtocol,
                     char          *         graphName)
```

Description

The nvotGetVerticesInGraph routine issues a get operation of all vertices contained in the graph identified by graphProtocol and graphName.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocol and graphName parameters are required. If one of these parameters is not provided, a vertex list is not searched for and the error code NVOT_GRAPH_INVALID_INDEX is set.

If the containing graph does not exist, the routine does not search for the vertices and the error code NVOT_GRAPH_DOES_NOT_EXIST is set.

If the get operation fails, the routine returns NULL and the error variable is set. For more information, see the following error codes and return values sections.

If the get operation completes successfully but no vertex exists within the graph, the routine returns NULL but the error variable is set to NVOT_SUCCESS.

If the get operation completes successfully and vertices exist in the graph, the routine returns a list of all vertices contained in the identified graph.

The interface allocates structured data in memory and returns a pointer to it. The user must call one of the free memory routines to have all data de-allocated.

Parameters

graphProtocol Specifies the protocol of the graph this routine looks at. This is the graph for whose vertices this routine searches. For more information, refer to the file /usr/OV/conf/oid_to_protocol.

graphName Specifies the name of the graph this routine looks at. Both the graphName and graphProtocol parameters are required to identify the containing graph. This parameter can be any string of characters. Once specified, the same name must be used in any reference to this graph.

Return Values

nvotVertexList Upon completion of the get operation, this routine returns a list of all vertices that are members of the identified graph.

As defined in the file nvotTypes.h, nvotVertexList is an array list of nvotVertex structures. The nvotVertexList return value is a structure made up of a pointer to the first nvotVertex element and an integer variable count indicating the number of elements in the list.

In its turn, each nvotVertex element is a structure carrying the actual information about a vertex. For more information, see Chapter 6, “Using NetView for AIX GTM Data Structures” on page 1061.

The variable operation returned in each nvotVertex element has no meaning in this routine.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_ERROR_ALLOCATING_MEMORY] Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example checks all vertices contained in the graph created in the example in “nvotCreateVertexInGraph(3)” on page 265.

```
#include <nvot.h>

nvotReturnCode         rc;
nvotVertexList    *    myVertices = NULL;

nvotGraphProtocolType  myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char         *         myGraphName = "My_STARLAN_Graph";

  if ((myVertices = nvotGetVerticesInGraph (myGraphProt,
                                            myGraphName) != NULL) {

    /* OK, it seems we have gotten a few vertices. Print their names. */
    printf ("Graph %s contains %d vertices.\n", myGraphName, myVertices->count);
```

```
    for (i = 1; i = myVertices->count; i++)
      printf ("Vertex %d=%s.\n", i, myVertices->vertex[i-1].vertexAttr.vertexName);

    /* We don't need them any longer. Let's release all memory. */
    nvotFreeVertexList (myVertices);

  } else {
    /* No, we've gotten no vertices. What happened?... */
    if ((rc = nvotGetError()) EQ NVOT_SUCCESS)
      printf ("Graph %s contains no vertices.\n", myGraphName);
    else {
      printf ("Error occurred getting vertices from graph %s\n", myGraphName);
      printf ("Error message : %s.\n", nvotGetErrorMsg (rc));
    }
  }
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotFree(3)” on page 304.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotInit(3)

Purpose

Initializes the NetView for AIX Open Topology interface

Syntax

```
#include <nvot.h>
nvotReturnCode  nvotInit (
                    char *           hostname,
                    nvotBooleanType  arcDirection)
                    nvotBooleanType  checkOn)
```

Description

The main purpose of this routine is to open a connection between an application and gtmd, either locally or remotely. The application using the NetView for AIX Open Topology platform can be running in one machine while gtmd is be running on another machine.

Hostname is the name of the network node where the gtmd is running. Hostname serves remote connections only. When an application and gtmd are to be running in the same machine, the hostname parameter should be set to NULL. Communication between the application and gtmd is based on socket connection. Although it is possible to connect a locally running application through an Internet type socket, it is less efficient than through a Unix type socket.

The nvotInit routine also determines how an interface handles arc objects and affects gtmd's ability to recover from lost traps.

An arc object is referred to by two endpoints - aEndpoint and zEndpoint. When an object is created in gtmd, it is given a name that must be used in any further references to it. However, the name of an arc object is derived from the names of two other objects, vertices and/or graphs. For example, an arc created as ***aEndpoint, zEndpoint, arcIndexId*** can be referred to as ***zEndpoint, aEndpoint, arcIndexId***. The arcDirection parameter tells the gtmd interface to initialize itself to handle arcs in one of the following ways:

- A value of TRUE notifies the interface that the names `aEndpoint, zEndpoint, 1` and `zEndpoint, aEndpoint, 1` refer to the same arc. In this case, additional processing is required for the interface to handle the arcs.
- A value of FALSE notifies the interface to handle such arcs as two distinct arcs.

In some situations, the gtmd can automatically recover from lost traps. For example, if a request is sent to create a vertex inside a graph, the containing graph must exist so that gtmd can associate the vertex it creates with the graph. If gtmd finds that the containing, or parent, graph does not exist, it determines that a previous trap sent to create that graph has been lost in the network. Then, it creates a default graph and associates the vertex with it. A default graph has its graphType attribute set to GRAPH and its layoutAlgorithm set to NONE_LAYOUT. These attributes are not allowed to be changed in the future.

The nvotInit routine's parameter variable ***checkOn*** specifies whether or not a function call causes the interface to query gtmd to determine whether a graph related to that call exists. If set to TRUE, any function call involving a parent graph causes the interface to query gtmd for the existence of that graph. If set to FALSE, the interface does not issue any queries about graphs.

The nvotInit routine can be used multiple times to re-establish a connection with gtmd. If the interface ever returns the message NVOT_GTMD_CONNECTION_ERROR and NVOT_SOCKET_ERROR, the nvotInit routine can be called and the connection to gtmd re-established without the application being restarted. But the nvotInit routine can be used only once in a session to set the arcDirection option. That is, in subsequent calls to nvotInit, the value of arcDirection must be the same as that in the first call.

Parameters

hostname A string of characters containing the hostname of the network node that is running the gtmd with which this application will connect.

arcDirection A boolean type parameter that determines how the interface should handle the direction of the arc object. If initialized with TRUE, the interface treats two arc names with the same endpoints in different order as the same arc, and determines the direction from one of the arc names. If initialized with FALSE, the interface treats two arc names with the same endpoints in different order as two different arcs, and determines the direction of the arcs as it receives them.

checkOn A boolean type parameter that specifies whether or not an interface will query the gtmd for the existence of graphs related to function calls when it receives a function call. When set to TRUE, the interface queries gtmd for the existence of the graphs involved in the function call. This prevents the interface from automatically creating a graph when one already exists. When set to FALSE, the interface does not query gtmd for the existence of graphs.

Return Values

If successful, nvotInit returns [NVOT_SUCCESS]. If unsuccessful, nvotInit returns one of the following error codes.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_ALREADY_INITIALIZED] Already initialized. The routine attempted to re-establish the connection with gtmd but the connection is still open.

[NVOT_GTMD_CONNECTION_ERROR]
There is a GTM connection error. The connection cannot be opened. Issue the nvotInit routine again.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the routine nvotGetErrorMsg, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT=SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a connection to gtmd and checks the result. The gtmd daemon should be running on the same host as the application.

The interface does not change the direction of the arcs. The existence of graphs is verified before objects are created.

```
#include <nvot.h>

nvotReturnCode          rc;
char              *     gtmHost = NULL;
nvotBooleanType         arcDirection = FALSE;
nvotBooleanType         graphQueryOn = TRUE;
  if ((rc = nvotInit (gtmHost, arcDirection, graphQueryOn)) == NVOT_SUCCESS)
    printf ("OK : %s\n", nvotGetErrorMsg (nvotGetError()));
  else
    printf ("WHOOPS! : %s\n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

When compiling a program that uses the nvotInit routine, you need to link to the following library:

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotDone(3)” on page 302.
- See “nvotGetError(3)” on page 323.
- See “nvotGetErrorMsg(3)” on page 326.

nvotSetCenterBoxForGraph(3)

Purpose

Specifies what box symbol is to be the center of a star graph map

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotSetCenterBoxForGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  centerBoxProtocol,
                         char         *         centerBoxName)
```

Description

The nvotSetCenterBoxForGraph routine sets up the symbol associated with the box graph identified by centerBoxProtocol and centerBoxName to be in the center of a STAR_LAYOUT graph identified by graphProtocolParent and graphNameParent.

The parent graph must have been created with layout algorithm set to STAR_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, centerBoxProtocol, and centerBoxName parameters are requierd. If one of these paraemters is not provided, the error code NVOT_GRAPH_INVALID_INDEX or NVOT_BOX_INVALID_INDEX is returned.

If the parent graph does not exist or it exists but its graphType is not set to GRAPH, the operation is rejected and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

If the box graph does not exist or it exists but its graphType is not set to BOX, the operation is rejected and the error code NVOT_BOX_DOES_NOT_EXIST is returned.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in the GTM database. This parameter is a string of characters used to create the parent graph.

centerBoxProtocol Specifies the protocol of the box graph whose symbol is to be in the center of the star map. For more information about specifying a box graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

centerBoxName Specifies the name of the center box graph. Both the centerBoxName and centerBoxProtocol parameters are required to identify the center box graph. This parameter is a string of characters used to create the box graph.

Return Values

nvotReturnCode The nvotSetCenterBoxForGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS] Successful operation.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol and/or name must not be NULL.

[NVOT_GRAPH_DOES_NOT_EXIST] A graph does not exist in the GTM database.

[NVOT_BOX_INVALID_INDEX] The box index is not valid. A box graph protocol and/or name must not be NULL.

[NVOT_BOX_DOES_NOT_EXIST] The box graph does not exist in the GTM database.

[NVOT_GTMD_INVALID_RESPONSE] The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example assumes that you have created a graph with layout=STAR_LAYOUT and containing several boxes. This example will cause the view to be redrawn so that *my_STARLAN_BoxName* would appear in the center.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  my_STARLAN_GraphsProt = "1.3.6.1.2.1.2.2.1.3.11";

char         *         myRoot_STARLAN_GraphName = "My_Root_Graph";
char         *         my_STARLAN_BoxName = "My_Box_STARLAN_Graph";

  if ((rc = nvotSetCenterBoxForGraph (my_STARLAN_GraphsProt,
                                      myRoot_STARLAN_GraphName,
                                      my_STARLAN_GraphsProt,
                                      my_STARLAN_BoxName)) == NVOT_SUCCESS)

    printf ("%s is the center of graph %s.\n", my_STARLAN_BoxName,
                                                  myRoot_STARLAN_GraphName);
  else
    printf ("An error occurred changing %s symbol position.\n",
                                                    my_STARLAN_BoxName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateBoxInGraph(3)” on page 227.
- See “nvotChangeBoxPositionInGraph(3)” on page 166.

nvotSetCenterGraphForGraph(3)

Purpose

Specifies what graph symbol is to be the center of a star graph map

Syntax

```
#include <nvot.h>

nvotReturnCode     nvotSetCenterGraphForGraph (
                         nvotGraphProtocolType  graphProtocolParent,
                         char         *         graphNameParent,
                         nvotGraphProtocolType  centerGraphProtocol,
                         char         *         centerGraphName)
```

Description

The nvotSetCenterGraphForGraph routine sets up the symbol associated with the graph identified by centerGraphProtocol and centerGraphName to be in the center of a STAR_LAYOUT graph identified by graphProtocolParent and graphNameParent.

The parent graph must have been created with layout algorithm set to STAR_LAYOUT.

The protocol and name parameters uniquely identify objects in the GTM database. The graphProtocolParent, graphNameParent, centerGraphProtocol, and centerGraphName parameters are required. If one of these parameters is not provided, the error code NVOT_GRAPH_INVALID_INDEX is returned.

If the parent graph or the center graph does not exist or they exist but their graphType attributes are not set to GRAPH, the operation is rejected and the error code NVOT_GRAPH_DOES_NOT_EXIST is returned.

Parameters

graphProtocolParent Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

graphNameParent Specifies the name of the containing graph. Both the graphNameParent and graphProtocolParent parameters are required to uniquely identify the parent graph in the GTM database. This parameter is a string of characters used to create the parent graph.

centerGraphProtocol Specifies the protocol of the graph whose symbol is to be in the center of the star map. For more information about specifying a graph's protocol, refer to the file /usr/OV/conf/oid_to_protocol.

centerGraphName Specifies the name of the center graph. Both the centerGraphName and centerGraphProtocol parameters are required to uniquely identify the center graph. This parameter is a string of characters used to create the graph.

Return Values

nvotReturnCode The nvotSetCenterGraphForGraph routine returns an nvotReturnCode that can assume the values described in the following error codes section.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_GRAPH_INVALID_INDEX]	The graph index is not valid. A graph protocol and/or name must not be NULL.
[NVOT_GRAPH_DOES_NOT_EXIST]	A graph does not exist in the GTM database.
[NVOT_GTMD_INVALID_RESPONSE]	The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.
[NVOT_ERROR_ALLOCATING_MEMORY]	Memory allocation error. The system might be out of memory.
[NVOT_NOT_INITIALIZED]	Not initialized. Issue the nvotInit routine to establish a connection with gtmd.
[NVOT_SOCKET_ERROR]	There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example assumes that you have created a graph with layout=STAR_LAYOUT and containing several graphs. This example will cause the view to be redrawn so that *myGraphName* would appear in the center.

```
#include <nvot.h>

nvotReturnCode         rc;

nvotGraphProtocolType  mySDLCGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char         *         myRootSDLCGraphName = "My_Root_Graph";
char         *         myGraphName = "My_Graph";

  if ((rc = nvotSetCenterGraphForGraph (mySDLCGraphsProt,
                                        myRootSDLCGraphName,
                                        mySDLCGraphsProt,
                                        myGraphName)) == NVOT_SUCCESS)

    printf ("%s is the center of graph %s.\n", myGraphName,
                                                    myRootSDLCGraphName);
  else
    printf ("An error occurred changing %s symbol position.\n", myGraphName);
  printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotCreateRootGraph(3)” on page 249.
- See “nvotCreateGraphInGraph(3)” on page 235.
- See “nvotChangeGraphPositionInGraph(3)” on page 184.

nvotSetSynchronousCreation(3)

Purpose

Initializes the interface to return OVwObjectIds

Syntax

```
#include <nvot.h>
nvotReturnCode  nvotSetSynchronousCreation (unsigned  synchronous)
```

Description

Objects stored in the gtmd database are also stored in the NetView for AIX object database. Objects in the object database are identified by an OVwObjectId. Although the interface is not meant to handle objects in the object database, the application can request that the interface creation functions return OVwObjectIds.

The nvotSetSynchronousCreation routine specifies whether the interface returns OVwObjectIds upon return from the creation function calls. The ***synchronous*** parameter is a timeout value, that specifies the number of seconds for the interface to wait for the OVwObjectId. If ***synchronous*** is set to a positive value, a connection is opened with the NetView for AIX object database, and the creation function calls return OVwObjectIds. If ***synchronous*** is set to 0 (zero) or nvotSetSynchronousCreation is not called at all, no connection is opened with the object database, and the creation function calls do not return OVwObjectIds. The interface create functions run faster in this case.

Parameters

synchronous
: An unsigned number, which determines the number of seconds the interface should wait for OVwObjectIds to be returned on creation routines. If this parameter is set to zero OVwObjectIds will not be returned.

Return Values

If successful, nvotSetSynchronousCreation returns [NVOT_SUCCESS]. If unsuccessful, nvotSetSynchronousCreation returns one of the following error codes.

Error Codes

[NVOT_SUCCESS]
: Successful operation.

[NVOT_OVW_CONNECTION_ERROR]
: There is an connection error with the NetView for AIX program. The connection to the NetView for AIX object database cannot be opened.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```
nvotReturnCode  rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf (“%s\n”, nvotGetErrorMsg (rc));
```

Examples

The following example creates a connection to the NetView for AIX object database. After this call, every create function should return the OVwObjectId of the created object.

```
#include <nvot.h>

nvotReturnCode          rc;
int                     timeout = 20;

  if ((rc = nvotSetSynchronousCreation (timeout)) == NVOT_SUCCESS)
    printf ("OK : %s\n", nvotGetErrorMsg (rc));
  else
    printf ("WHOOPS! : %s\n", nvotGetErrorMsg (rc));
```

Libraries

- /usr/OV/lib/libnvot.a

Files

nvot.h

Related Information

- See “nvotInit(3)” on page 359.

nvotVertexHandler(3)

Purpose

Provides open access to all tables, variables, and operations defined in the NetView for AIX Generic Topology MIB.

Related Functions

nvotGraphHandler
nvotArcHandler
nvotSapHandler
nvotSimpleConnectionHandler
nvotUnderlyingConnectionHandler
nvotUnderlyingArcHandler
nvotMemberHandler
nvotMemberArcHandler
nvotAttachedArcHandler
nvotAdditionalMemberHandler
nvotAdditionalGraphHandler

Syntax

```
#include <nvot.h>

nvotReturnCode  nvotVertexHandler (nvotVertex * vertex)

nvotReturnCode  nvotGraphHandler (nvotGraph * graph)

nvotReturnCode  nvotArcHandler (nvotArc * arc)

nvotReturnCode  nvotSapHandler (nvotSap * sap)

nvotReturnCode  nvotSimpleConnectionHandler
                        (nvotSimpleConnection * simpleConnection)

nvotReturnCode  nvotUnderlyingConnectionHandler
                        (nvotUnderlyingConnection * underlyingConnection)

nvotReturnCode  nvotUnderlyingArcHandler (nvotUnderlyingArc * underlyingArc)

nvotReturnCode  nvotMemberHandler (nvotMember * member)

nvotReturnCode  nvotMemberArcHandler (nvotMemberArc * memberArc)

nvotReturnCode  nvotAttachedArcHandler (nvotAttachedArc * attachedArc)

nvotReturnCode  nvotAdditionalMemberHandler
                            (nvotAdditionalMember * additionalMember)

nvotReturnCode  nvotAdditionalGraphHandler
                            (nvotAdditionalGraph * additionalGraph)
```

Description

Basic routines have been made available for those programmers with more familiarity with NetView for AIX Generic Topology MIB, its manageable resources, tables, attributes and operations.

While the convenience routines are intended to ease NetView for AIX Topology Application programming, through the encapsulation of sets of lower-level functions into higher level calls to the GTM interface, the Handler routines allow for complete open access to all tables, variables and operations defined in NetView for AIX Generic Topology MIB through the free combination of lower-level functions.

Unlike the convenience routines these basic lower-level routines leave to the programmer the burden of certain checks and controls. For example, even if the application programmer wants to avoid automatic creation of graphs when creating objects in graph submaps, it is his task to make sure the parent graph exists. When deleting a graph, he must make sure it contains no objects inside. Even when displaying an object in OVW, the programmer must make sure that the object is a member of a graph which in turn belongs to a hierarchy of a root graph.

For example, suppose you want to create a graph G2 to be displayed in the submap of root graph G1 that does not exist yet. Using the Convenience routines the task could be accomplished through the following steps:

- Call nvotCreateGraphInGraph (G1, G2)
- If the creation above returns NVOT_GRAPH_DOES_NOT_EXIST, create graph G1 and call again nvotCreateGraphInGraph (G1, G2).

Now, suppose you want to create 10 vertices inside graph G2 above. Because convenience routine nvotCreateVertexInGraph is actually issuing a get graph for every vertex creation and we know that graph G2 already exists, it might be advantageous to performance to accomplish the task using these basic routines instead. Pseudocode for this would be as follows:

- Issue a get to graph G2 (e.g., nvotGetGraphsInGraph (G1) or nvotGetGraphObjectId (G2))
- If graph (G2) exists
- For (n = 1; n = 10; n++)
- If (nvotVertexHandler (CREATE_OPERATION, Vn) NE NVOT_SUCCESS)
- Break

However, when handling a smaller number of objects into different submaps, it is preferable to use the convenience routines and let the GTM interface take care of all the necessary checks.

The convenience routines and the basic routines can be mixed within a program.

Aside from those restrictions implied by the NetView for AIX Generic Topology MIB itself, all tables, variables and operations are available through the Basic Routines. However, it is recommended that the programmer be familiar with the results yielded by each action.

Each one of these basic routines is targeted to one individual table, or object, defined in NetView for AIX Generic Topology MIB. In other words, each basic routine handles one specific table. Each routine takes as parameter a pointer to a structure into which are mapped all table variables or attributes of a corresponding object.

In addition to the object attributes two other variables are present in all structures - a *nvotOperationType* and a *nvotAttributeBitmapType* variable.

nvotOperationType is set to the operation to be executed upon the table. The following operations can be performed:

- New: Create a new object or relationship
- Delete: Delete an object or relationship
- Variable Value Change: Change one or more variables
- Operational State Change: Change one or more status variables

nvotAttributeBitmapType is a bitmask indicating which attributes the application wants to operate upon. If a bit is set, the corresponding variable must be assigned a valid value, especially those variables of type (char *). Be aware that certain variables cannot be validated, such as char pointer type (which must be set to a valid string if its corresponding *validAttributes* bit is on).

In addition, a *nvotNamebindingType* is present in all structures except nvotVertex, nvotGraph, nvotBox, nvotSap, nvotMember, and nvotAdditionalGraph. nvotNamebindingType supplies more specific information regarding the object index such as if an arc is identified by its endpoints. Since the endpoints of an arc may be vertices, graphs or a vertex and a graph *nvotNamebindingType* variable specifies each endpoint. Note that the *nvotNamebindingType* variable must be assigned a value that matches the index variables to which it is related. Similarly, in the case of an arc structure, if *nameBinding* is assigned a value of ARC_GRAPH_VERTEX_NAME_BINDING, this incates that the aEndpoint index fields must be set with a graph index and the zEndpoint index fields must be set with a vertex index. This is true for all structures using a *nvotNamebindingType* variable.

As soon as a basic structure is filled out appropriately, a call to the related basic routine accomplishes the operation in the NetView for AIX topology database and possibly in a topology map in the OVw display.

Upon completion a return code is available.

Note that no dependency checks are applied upon the operations. For example, suppose a call is made to the ***nvotMemberHandler*** routine with the purpose of associating vertex V1 to graph G1. The ***nvotMembers*** structure might be set as follows:(as illustration purpose only) :

```
struct {
         operation        = CREATE_OPERATION;
         validAttributes  = MEMBERPROTOCOL_ATTR BITOR MEMBERNAME_ATTR BITOR
                            MEMBERNAMEBINDING_ATTR BITOR
                            MEMBERCOMPONENTPROTOCOL_ATTR BITOR
                            MEMBERCOMPONENTNAME_ATTR;
         struct {
                  memberProtocol          = "1.3.6.1.2.1.2.2.1.3.11";
                  memberName              = "G1_Name;
                  nameBinding             = VERTEX_NAME_BINDING;
                  memberComponentProtocol = STARLAN;
                  memberComponentName     = "V1_Name";
                  memberLabel             = NULL;
                  memberIcon              = NULL;
         } nvotMembersAttrType;
} nvotMembers;
```

This operation will not verify whether graph G1 or vertex V1 exists. NetView for AIX attempts to perform the operation. If either of the objects does not exist they will be automatically created with default values by the GTM recovery process.

The operation above could also be accomplished through the convenience routines by calling ***nvotCreateVertexInGraph*** or ***nvotCreateVertexInBox***.

Parameters

Each basic routine takes a pointer to a basic structure which is associated to a NetView for AIX Generic Topology MIB table.

vertex
A pointer to a *nvotVertex* type structure defined for the handling of the Vertex table.

graph
A pointer to a *nvotGraph* type structure defined for the handling of the Graph table. *nvotGraph* is also used for the handling of box since this one is a graph with graphType attibute set to BOX.

arc
A pointer to a *nvotArc* type structure defined for the handling of the Arc table.

sap
A pointer to a *nvotSap* type structure defined for the handling of the Sap table.

simpleConnection
A pointer to a *nvotSimpleConnection* type structure defined for the handling of the SimpleConnection table.

underlyingConnetion
A pointer to a *nvotUnderlyingConnection* type structure defined for the handling of the UnderlyingConnection table.

underlyingArc
A pointer to a *nvotUnderlyingArc* type structure defined for the handling of the UnderlyingArc table.

member
A pointer to a *nvotMember* type structure defined for the handling of the Member table.

memberArc
A pointer to a *nvotMemberArc* type structure defined for the handling of the MemberArc table.

attachedArc
A pointer to a *nvotAttachedArc* type structure defined for the handling of the AttachedArc table.

additionalMember
A pointer to a *nvotAdditionalMember* type structure defined for the handling of the Member additional information table.

additionalGraph
A pointer to a *nvotAdditionalGraph* type structure defined for the handling of the Graph additional information table.

Return Values

nvotReturnCode	Basic routines will return an nvotReturnCode that can assume the values described in the item Error Codes bellow.

Error Codes

[NVOT_SUCCESS]	Successful operation.
[NVOT_INVALID_OPERATION]	The basic routines do not support a GET operation.
[NVOT_INVALID_NAME_BINDING]	The name binding is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_NULL_BASIC_STRUCT_POINTER]
The basic routine has been called with a NULL pointer to a basic structure.

[NVOT_VERTEX_INVALID_INDEX] The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

[NVOT_GRAPH_INVALID_INDEX] The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_ENDPOINT_INVALID_INDEX] The graph endpoint index is not valid.

[NVOT_ENDPOINT_GRAPH_INVALID_INDEX]
The endpoint graph index is not valid. An endpoint graph protocol or name must not be NULL.

[NVOT_SAP_INVALID_INDEX] The SAP index is not valid. A SAP protocol must be a positive integer and a SAP name must not be NULL.

[NVOT_INVALID_INDEX_VALUE] Could not determine which index is valid.

[NVOT_INVALID_STATUS] The status is not valid. It must be a number defined in the nvotTypes.h file.

[NVOT_ERROR_ALLOCATING_MEMORY]
Memory allocation error. The system might be out of memory.

[NVOT_SESSION_TO_GTM_NOT_ESTABLISH]
The routine attempted to establish connection with gtmd but the connection has not been opened yet.

[NVOT_NOT_INITIALIZED] Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

[NVOT_SOCKET_ERROR] There is a socket error. The connection failed during operation. Issue the nvotInit routine again.

A printable message string is accesseable through a call to the routine nvotGetErrorMsg as in the example bellow:

```
nvotReturnCode  rc;

if (rc != NVOT_SUCCESS)
  printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

See the /usr/OV/prg_samples/nvot directory for an example of how to construct a complete topology using the Basic routines. It is in the file *nvotBasicSerialUnderlyingArc.c*.

The following code fragment shows an example from this sample program. This code defines a graph structure to be used with the nvotGraphHandler basic routine.

```
/*********************************************************************/
/* Creating components                                               */
/*********************************************************************/

/*********************************************************************/
/* Root graph                                                        */
/*********************************************************************/

/************************
 ** GRAPH OPERATION - Filling nvotGraph Structure
 ************************/
   operation = CREATE_OPERATION ;
   graph.operation = operation ;
   graph.validAttributes = 0 ;
   graph.graphAttr.graphType = GRAPH ; /* 3 */
   graph.validAttributes |= GRAPHTYPE_ATTR ;

   graph.graphAttr.graphProtocol = strdup( "1.3.6.1.2.1.2.2.1.3.2" ) ;
   graph.validAttributes |= GRAPHPROTOCOL_ATTR ;

   graph.graphAttr.graphName = strdup( "Serial_Underlying_Arc_Basic_Root" ) ;
   graph.validAttributes |= GRAPHNAME_ATTR ;

   graph.graphAttr.layoutAlgorithm = POINT_TO_POINT_LAYOUT ; /* 3 */
   graph.validAttributes |= LAYOUTALGORITHM_ATTR ;

   graph.graphAttr.isRoot = TRUE ;
   graph.validAttributes |= ISROOT_ATTR ;

   rc = nvotGraphHandler( NetView for AIX Grapher );

   nvotFreeGraph (&graph);
```

Libraries

libnvot.a

Files

- nvot.h

nvSnmpBlockingGetTable(3)

Purpose

Retrieves an entire table from the MIB or an element from the returned table

Related Functions

nvSnmpGetTableElement
nvSnmpGetTable
nvSnmpXGetTable

Syntax

```
# include <OV/OVsnmp.h>

OVsnmpPdu *nvSnmpBlockingGetTable(OVsnmpSession *session,
       ObjectID *oid, int oidLen, int *rows, int *columns);

int nvSnmpGetTable(OVsnmpSession *session,
       ObjectID *oid, int oidLen, int *rows, int *columns);

int nvSnmpXGetTable(OvsnmpSession *session,
       ObjectID *oid, int oidLen, int *rows, int *columns);

OVsnmpVarBind *nvSnmpGetTableElement (OVsnmpPdu *pduPtr,
       int totalRows, int row, int column);
```

Description

The nvSnmpBlockingGetTable, nvSnmpGetTable, and nvSnmpXGetTable calls retrieve an entire table from the MIB. These convenience calls prevent you from having to write the code to retrieve entire tables. The convenience function, nvSnmpGetTableElement, enables the user to retrieve an element from the returned table.

The nvSnmpBlockingGetTable call retrieves an entire table in a blocking manner. The user passes to the call a session that has been established with the OVsnmpOpen call, the object ID of the table to be retrieved (the oid immediately before the first instance in the table), the length of the object ID, and a pointer to two integers in which the number of rows and columns in the requested table are returned. For example, if you want to retrieve the interface table, pass the oid associated with ifEntry, for example, 1.3.6.1.2.1.2.2.1. Sample code is provided in the file /usr/OV/prg_samples/nvsnmp_app/getTable.c.

You should use the OVsnmpApi.h header file with the nvSnmpBlockingGetTable call. This header file provides the definition for the ObjectID type.

The nvSnmpGetTable call has the same parameters as the blocking version. However, this call is made in a non-blocking manner. When using this call, you must establish a select statement to manage the transmissions.

Note: The nvSnmpGetTableElement call does not provide any check on the tuple (row, column) passed as parameter. You must check the accuracy of these parameters.

The nvSnmpXGetTable also has the same parameters as nvSnmpBlockingGetTable and nvSnmpGetTable. However, if you use nvSnmpXGetTable, you must use the OVsnmpXOpen to manage the transmissions.

Parameters

session	Specifies a pointer to a session that was previously established by the application
oid	Specifies a pointer to the table object identifier
oidLen	Specifies the length of the table oid
rows	Specifies the address of an integer that will return the number of rows in the retrieved table
columns	Specifies the address of an integer that receives the number of columns in a retrieved table
pduPtr	Specifies a pointer to a received PDU
totalRows	Specifies the total number of rows in the table
row	Specifies the row from which a value should be retrieved from the table
column	Specifies the line from which a value should be retrieved from a table

Return Values

If successful, nvSnmpBlockingGetTable returns a pointer to an OVsnmpPdu structure. If unsuccessful, it returns NULL. If successful, nvSnmpGetTable and nvSnmpXGetTable return a request ID. If unsuccessful, they return –1 (negative one). If successful, nvSnmpGetTableElement returns an OVsnmpVarBind pointer. If nvSnmpGetTableElement is unsuccessful and the PDU sent is not corrupted, the call returns NULL.

Error Codes

The nvSnmpBlockingGetTable call returns the error code value OVsnmpErrno. The following list describes the possible errors:

[SNMP_ERR_NO_RESPONSE]	No response received before a time-out occurred.
[SNMP_ERR_BAD_SESSION]	The session parameter does not point to an OVsnmpSession data structure that was created by OVsnmpOpen.
[SNMP_ERR_PDU_BUILD]	An internal error occurred while ASN.1 encoding the PDU. One of the variables might not have a valid type. This can happen if the OVsnmpVarBind data structure is modified after a call to OVsnmpAddTypedVarBind.
[SNMP_ERR_BAD_PDU_TYPE]	The OVsnmpPdu data structure was not a get request, get next request, or a set request. This may happen if the OVsnmpPdu data structure is modified after a call to OVsnmpCreatePdu.
[SNMP_SYSERR_SENDTO]	The sendto system call failed. The external variable errno contains the sendto specific error.
[SNMP_SYSERR_SELECT]	The select system call failed. The external variable errno contains the select specific error.
[SNMP_SYSERR_MALLOC]	The malloc system call failed. The external variable errno contains the malloc specific error.

Examples

See the /usr/OV/prg_samples/nvsnmp_app/getTable.c file.

Implementation Specifics

See “OVsnmpSend(3)” on page 514 for specific information about using select to wait for the response to arrive.

Libraries

When compiling a program that uses nvSnmpBlockingGetTable or one of its related calls, you need to link to the following libraries:

- /usr/OV/lib/libov.a
- /usr/OV/lib/libnvsnmp.a
- /usr/OV/lib/libovc.a
- /usr/OV/lib/libovcmapi.a
- /usr/OV/lib/libntl.a

The library nvsnmp replaces the original ovsnmp library. You must link to the nvsnmp library to use the extended APIs and you **must not** simultaneously use the original ovsnmp library. Other OVsnmp APIs are replicated in nvsnmp, even though they are not related to filtering facilities.

Files

/usr/OV/include/OV/OVsnmp.h

nvSnmpErrString(3)

Purpose

Returns SNMP specific error strings

Syntax

```
#include <OV/OVsnmp.h>

    char * nvSnmpErrString(int nvsnmpSubsys,
                           int nvsnmpErrno);
```

Description

The nvSnmpErrString call returns a textual string that provides information about the error specified in the nvsnmpErrno parameter. Use this routine with nvSnmpTrapOpenFilter to enable your application to identify errors caused by malfunctions of the communications infrastructure.

Return Values

The nvSnmpErrString call returns a pointer to a static character string. This string is read only. If the error number is out of range,the nvSnmpErrString call returns the string Unknown Error.

Libraries

When compiling a program that uses the nvSnmpErrString call, link to the following libraries:

- /usr/OV/lib/libov.a
- /usr/OV/lib/libnvsnmp.a
- /usr/OV/lib/libovc.a
- /usr/OV/lib/libovcmapi.a
- /usr/OV/lib/libntl.a

The library nvsnmp replaces the original ovsnmp library. You must link to nvsnmp to use the extended APIs and **must not** simultaneously use the original ovsnmp library. Other OVsnmp APIs are replicated in nvsnmp, even though they are not related to filtering facilities.

Files

/usr/OV/include/OV/OVsnmp.h

Related Information

- See “nvSnmpTrapOpenFilter(3)” on page 380.

nvSnmpTrapOpenFilter(3)

Purpose

Opens a session with event management services (EMS) to receive SNMP filtered traps

Related Functions

nvSnmpXTrapOpenFilter

Syntax

```
#include <OV/OVsnmp.h>
#include <OV/OVsnmpXfns.h>
/* This second include statement applies only to nvSnmpXTrapOpenFilter */

OVsnmpSession *nvSnmpTrapOpenFilter( void (*callback)(),
       void *data, char *filter );

OVsnmpSession *nvSnmpXTrapOpenFilter(XtAppContext context, void (*callback)(),
       void *data, char *filter);
```

Dependencies

These functions APIs depend on the EMS (ovesmd) daemon. If the EMS daemon is not running, nvSnmpTrapOpenFilter fails.

Description

The nvSnmpTrapOpenFilter and nvSnmpXTrapOpenFilter APIs enable registration to receive filtered traps. The usage of these APIs is similar to that of OVsnmpTrapOpen and OVsnmpXTrapOpen. The application opens a session to receive traps, passing a callback routine to be called when the trap arrives, and optional data. To use the filter facilities, the application must pass a NULL-terminated string representing a filter. If you do not want any filtering, the application must pass a NULL filter parameter. The other parameters are the same as those used on OVsnmp APIs.

Filters are defined with the syntax defined under Keyword Syntax on page 381. Filters can be created from the Filter Editor and accessed with the nvFilter routines. Filters are represented by NULL-terminated strings with a defined syntax. Keywords are provided to determine the type of filtering, and to allow filters to be combined and compared to received events.

In using filtering facilities, an application may follow one of two approaches:

- Open as many sessions as the number of filters to be registered to receive traps. In this case, the application must handle multiple sessions to gather events.
- Open only one session that combines all filters. In this case, the application must build the filter string with the composition of all filters that need to be registered. To do this, the application must create a string ORing the rules to be registered, following the syntax defined under Keyword Syntax on page 381.

The callback has the following prototype:

```
void callback(int            type,
              OVsnmpSession  *session,
              OVsnmpPdu      *response,
              void           *userData);
```

Parameters

*OVsnmpSession * session*	Specifies a pointer to the session to be opened to receive traps.
*void (*callback)()*	Specifies the function that will be called to process a trap if the OVsnmpRead function is used to receive the trap. If the calling process wants to use callback procedures, this parameter must point to a valid function.
*void * data*	Specifies a pointer to application-specific data that will be passed to the callback function. No action is performed on this data.
*char * filter*	Specifies a pointer to a string that defines a filter.
XtAppContext context	Specifies the X application context.

Keyword Syntax

! NOT (logical negation)

&& AND (logical and)

|| OR (logical or)

The following list describes the keywords in the syntax used to define filters.

CLASS=value
SNMP enterprise match on enterprise ID. Value is given in dot notation, for example, 1.2.3.4.55

IP_ADDR=value
SNMP agent-addr match on IP address. Value is given in dot notation, for example, 192.155.13.57. Registration for an IP_ADDR permits receipt of agent-generated traps, as well as internal events related to that IP_ADDR.

LOGGED_TIME <= time_string
Time that was logged before the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

LOGGED_TIME >= time_string
Time that was logged after the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

PRESENT = SNMP_TRAP
Presence of SNMP Trap

SNMP_TRAP=value
Match on SNMP Generic Trap Type, where the Generic Type is an integer

SNMP_SPECIFIC=value
Match on SNMP Specific Trap Type, where the Specific Type is an integer

TIME_PERIOD=time_constant
Relative time period (integer seconds) for frequency filters

THRESHOLD <= frequency
Number of event occurrences is less than or equal to frequency (integer) during TIME_PERIOD

THRESHOLD >= frequency
Number of event occurrences is greater than or equal to frequency (integer) during TIME_PERIOD

Note: When included in an expression for nvSnmpTrapOpenFilter, the keywords THRESHOLD and TIME_PERIOD must be ANDed (never ORed) and grouped within parentheses as in the following example:

```
filter = PRESENT=SNMP_TRAP && (THRESHOLD <= 5 && TIME_PERIOD = 30)
```

Specifying more than 250 filter objects will result in an error.

Return Values

If successful, the nvSnmpTrapOpenFilter and nvSnmpXTrapOpenFilter calls return a pointer to a new OVsnmpSession structure. Memory allocated for the OVsnmpSession must be freed by OVsnmpClose. If unsuccessful, these calls return NULL.

If nvSnmpTrapOpenFilter and nvSnmpXTrapOpenFilter are unsuccessful, the external variables contain the following specific error values:

OVsnmpErrno	The SNMP specific error value
nvSnmpErrno	The specific errors of the nvSnmpTrapOpen call
nvSnmpSubsys	The subsystem where the error occurred

If the error code indicates a system call failure, the external variable *errno* will contain the corresponding system error values.

Error Codes

The nvSnmpTrapOpenFilter and nvSnmpXTrapOpenFilter calls return the error code value OVsnmpErrno, nvSnmpErrno, or nvSnmpSubsys. The following list describes the possible errors:

[SNMP_SYSERR_SOCKET]	A call to socket failed
[SNMP_SYSERR_MALLOC]	A call to malloc failed
[SNMP_SYSERR_BIND]	A call to bind
[NVSNMP_ERROR_OPEN]	An error opening a session
[NVSNMP_ERROR_CLOSE]	An error closing a session
[NVSNMP_ERROR_RECEIVE_PEEK]	An error peeking a message
[NVSNMP_ERROR_RECEIVE_PEEK_CNTL]	An error peeking a message
[NVSNMP_ERROR_RECEIVE_READ]	An error reading a message
[NVSNMP_ERROR_RECEIVE_READ_CNTL]	An error reading a message

[NVSNMP_ERROR_RECEIVE_CREATE]	Sieve creation failed; an error in receive confirmation
[NVSNMP_ERROR_RECEIVE_DELETE]	Sieve deletion failed; an error in receive confirmation
[NVSNMP_ERROR_RECEIVE_EVENT]	An error if the message received is not an event
[NVSNMP_ERROR_RECEIVE_TRAP]	An error if the message received is not a trap
[NVSNMP_ERROR_EVENT_IND]	An error decoding event indication
[NVSNMP_ERROR_CREATE_CNF]	An error receiving create sieve confirmation
[NVSNMP_ERROR_DELETE_CNF]	An error receiving delete sieve confirmation
[NVSNMP_ERROR_CREATE_SIEVE]	An error creating sieve on EMS
[NVSNMP_ERROR_DELETE_SIEVE]	An error deleting sieve on EMS
[NVSNMP_ERROR_DEFINE_FILTER]	An error in the filter defined as parameter
[NVSNMP_ERROR_EVENT_REGISTER]	An error registering for event reception
[NVSNMP_ERROR_EVENT_DEREGISTER]	An error deregistering for event reception
[NVSNMP_ERROR_TRAP_INFO]	An error in SNMP trap information; SNMP trap message received is empty
[NVSNMP_ERROR_CREATE_TIMEOUT]	Sieve creation failed; confirmation time-out
[NVSNMP_ERROR_DELETE_TIMEOUT]	Sieve deletion failed; confirmation time-out

Examples

See examples in /usr/OV/prg_samples/ovsnmp_app.

Libraries

When compiling a program that uses nvSnmpTrapOpenFilter or nvSnmpXTrapOpenFilter, link to the following libraries:

- /usr/OV/lib/libov.a
- /usr/OV/lib/libnvsnmp.a
- /usr/OV/lib/libovc.a
- /usr/OV/lib/libovcmapi.a
- /usr/OV/lib/libntl.a

The library nvsnmp replaces the original ovsnmp library. You must link to nvsnmp to use the extended APIs and **must not** simultaneously use the original ovsnmp library. Other OVsnmp APIs are replicated in nvsnmp, even though they are not related to filtering facilities.

Files

/usr/OV/include/OV/OVsnmp.h

Related Information

- See “OVsnmpTrapOpen(3)” on page 517

nvs_Audit(3)

Purpose

Defines format for audit entries to be entered in the security logfile

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/*The second include statement applies only if you want to parse error codes */

int nvs_Audit (int auditType, int InputCount, char *InputArray[])
```

Description

Use this function to define the format of entries to be added to the security logfile for audit purposes. The entry is defined in the form of an array.

Parameters

auditType	The type of audit information being collected (ATYPE_CONFIG).
InputCount	The number of elements in the array.
InputArray[]	An array of pointers. Each pointer points to the data to be put in the audit log.

Error Codes

If routine nvs_Audit is successful, it returns the string SEC_SUCCESS. Otherwise, it returns one of the following error codes:

[SEC_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[SEC_NOT_INITIALIZED]	The client did not initialize a security context with the security server.

Examples

The following example defines a 3-element array for an entry in the security logfile when a user unsuccessfully tries to modify the file name, file size, or port number of the trapd application.

```
char *InputArray[3];                    //ptr for each entry
⋮
// get storage for each buffer
⋮
char *object0="Trace filename ";  // field attempted to modify
char *value0="/tmp/trapd.filex";  //new value for the field
char *object1="File size";  //field attempted to modify
char *value1="100000";  //new falue for that field
char *object2="Port number";  //field attempted to modify
char *value2="888 888";   //new value for that field
sprintf(p[0], "%s %s, object0, value0); //contruct 1st entry
sprintf(p[1], "%s %s, object1, value1); //contruct 2nd entry
sprintf(p[2], "%s %s, object2, value2); //contruct 3rd entry

int nvs_Audit (ATYPE_CONFIG, 3, InputArray) ;
```

Libraries

When compiling a program that uses nvs_Audit, link to the following libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_Audit, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_deleteSecContext(3)” on page 386.
- See “nvs_getClientPerms(3)” on page 388.
- See “nvs_isClientAuthorized(3)” on page 391.

nvs_deleteSecContext(3)

Purpose

Closes a NetView for AIX client's security context with the NetView for AIX security server

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/* This second include statement applies only if you want to parse error codes */

int nvs_deleteSecContext
```

Description

This function is the means by which a NetView for AIX client closes its security context with the NetView for AIX security server. The gss_context handles are also released. This API *must* be called in your applications's exit handling routine. If security is off, a call to nvs_deleteSecContext returns a successful result.

Error Codes

If the nvs_deleteSecContext API function is successful, a result value of SEC_SUCCESS is set. Otherwise, one of the following error values is set:

[SEC_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[SEC_NOT_INITIALIZED] The client did not initialize a security context with the security server.

[SEC_DELETE_CONTEXT_FAILURE] The security context could not be deleted.

Examples

A typical call to this API at exit follows:

```
  main(){

  atexit(exitHandler)  //register exitHandler

}

static void exitHandler(){
nvs_deleteSecContext(); //close security context
other exit processing
}
```

Libraries

When compiling a program that uses nvs_deleteSecContext, link to the following libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_deleteSecContext, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_Audit(3)” on page 384.
- See “nvs_getClientPerms(3)” on page 388.
- See “nvs_isClientAuthorized(3)” on page 391.

nvs_getClientPerms(3)

Purpose

Obtains a bitmask representation of the permissions a user has for different NetView for AIX functions

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/* This second include statement applies only if you want to parse error codes */

void nvs_getClientPerms( char * tgtid, char * perms, int * status)
```

Description

This function is the means by which a NetView for AIX client can obtain the specific permissions a user has to NetView for AIX functions and objects. Instead of returning a SUCCESS or FAILURE code as in nvs_isClientAuthorized, an internal representation (bitmask) of the permissions associated with the target function is returned. By analyzing these bits, the caller can learn which permissions are granted and which are denied to the requesting user.

Bits 0-6 of the bitmask have predefined meanings as follows:

Table 15. Bitmask Permissions for nvs_getClientPerms API

Bit	Constant	Value	Meaning
0	sec_acl_perm_read	0*x*00000001	read
1	sec_acl_perm_write	0*x*00000002	write
2	sec_acl_perm_execute	0*x*00000004	execute
3	sec_acl_perm_control	0*x*00000008	control
4	sec_acl_perm_insert	0*x*00000010	insert
5	sec_acl_perm_delete	0*x*00000020	delete
6	sec_acl_perm_test	0*x*00000040	test

The semantics of bits 7-31 are defined by the application, and clients should contain code to handle them properly.

The user must have logged in to the NetView for AIX security server previously, using the authentication application.

Access control is effective only when global security is turned on. If security is off, all requests return an ALL_GRANTED result. All access control queries made through this API are automatically logged in the security audit files, although no indication of success or failure of the user's request is logged.

Parameters

tgtid Specifies the qualified name of the function or object for which permission information is being requested. For example, the function `SNMP MIB Browser->Tools->MIB Browser->SNMP` could be represented by the string `SNMP MIB Browser.Tools.MIB Browser.SNMP` This representation should match the one used in the domain profiles for the same entity.

perms Acts as a placeholder for the result of the query. Upon return, if the operation is successful (*status* variable set to SEC_SUCCESS), this argument will have the bitmask corresponding to the settings for permissions associated with *tgtid*, where 1 corresponds to granted permissions, and 0 corresponds to denied permissions.

If the operation is not successful, the value of this argument is not defined.

status Specifies a placeholder for the overall status of the call. Upon return, if the operation is successful, it contains the string SEC_SUCCESS. Otherwise, it contains one of the error codes listed below.

Error Codes

[SEC_CONNECTION_LOST]
The connection to the NetView for AIX program was lost.

[SEC_USER_NOT_LOGGED_ON]
The user who is requesting to use the function or object has not logged on through the NetView for AIX security server.

[SEC_USER_LOGINNAME_INVALID]
The NetView for AIX user ID specified is not authorized to perform the requested function.

[SEC_USER_GROUPNAME_INVALID]
The NetView for AIX group to which the specified NetView for AIX user ID belongs is not authorized to perform the requested function.

[SEC_USER_CLIENTNAME_INVALID]
The client machine name associated with the NetView for AIX user ID making the request is not a valid client for this security server.

[SEC_AUTHORIZATION_TARGETID_INVALID]
The target function or object for which authorization is being requested is invalid.

[SEC_AUTHORIZATION_TARGETPERM_INVALID]
The requested permission is not defined for the target string. This may indicate that the SRF and the security server are out of sync.

Examples

A typical call to this function follows:

```
  int st, permission ;

nvs_getClientPerms(“Edit->Add->Object,” &permission, &st )
if( st != SEC_SUCCESS
{
    Handle exception
    Interrupt flow of control (break, exit, return, ...)
}
if( ! (permission & sec_acl_perm_execute ) ) /* check for “x” perm */
[  Handle permission denial case
        Interrupt flow of control (break,exit,return, ...)
        Log “ACCESS FAILURE” by means of nvs_Audit API call
]
```

Libraries

When compiling a program that uses nvs_getClientPerms, link to the following shared libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_getClientPerms, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_Audit(3)” on page 384.
- See “nvs_deleteSecContext(3)” on page 386.
- See “nvs_isClientAuthorized(3)” on page 391.

nvs_isClientAuthorized(3)

Purpose

Queries a user's access to NetView for AIX functions to determine if a user can perform an action

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/* This second include statement applies only if you want to parse error codes */

void nvs_isClientAuthorized( char * tgtid, char * perms, int * authorized, int * status)
```

Description

This function returns information about a user's ability to access NetView for AIX functions and objects. If your application handles sensitive functions or objects, call this API to determine whether permission to use those functions or objects should be granted or denied to the user running the application. The user must have logged in to the NetView for AIX security server previously, using the authentication application.

Access control is effective only when NetView for AIX security is turned on. If security is off, all requests return a successful result. Use “nvs_isSecOn(3)” on page 395 to determine if NetView for AIX security is active.

Note: The nvs_isClientAuthorized function *must* be called to check a user's execute authority for an application at the beginning of the main() function. For example, for an application called `myapp`, use `myapp` as the tgtid and `x` (for execute) as the perm.

Parameters

tgtid Specifies the qualified name of the function or object for which access is being requested. For example, the function `View -> Automatic Layout -> For This Submap - > On For This Submap` can be represented by the string `View.Automatic Layout.For This Submap.On For This Submap`. This representation should match the one used in the domain profiles for the same entity.

perms Specifies a string of characters representing the desired level of permission to access the target. For example, `r` (for read) can be used to represent the level of permission needed to include a specific item in a menu, and `x` (for execute) can be used to represent the level of permission needed to enable the menu item. The letter `w` can be a generic indicator for the level of permission needed to update some target file or apply a change.

authorized Specifies a placeholder for the result of the query. Upon return, if the operation is successful (*status* variable contains SEC_SUCCESS), this argument has one of the two following values:

[SEC_AUTHORIZATION_SUCCESS] Permission was granted.
[SEC_AUTHORIZATION_FAILURE] Permission was denied.

If the operation is not successful, the value of this argument is not defined.

status Specifies a placeholder for the overall status of the call. Upon return, if the operation is successful, this variable contains the string SEC_SUCCESS. Otherwise, it contains an error code.

Error Codes

[SEC_CONNECTION_LOST]
The connection to the NetView for AIX program was lost.

[SEC_USER_NOT_LOGGED_ON]
The user who is requesting to use the function or object has not logged on through the NetView for AIX security server.

[SEC_USER_LOGINNAME_INVALID]
The NetView for AIX user ID is not authorized to perform the requested function.

[SEC_USER_GROUPNAME_INVALID]
The NetView for AIX group to which the specified NetView for AIX user ID belongs is not authorized to perform the requested function.

[SEC_USER_CLIENTNAME_INVALID]
The client machine name associated with the NetView for AIX user ID making the request is not a valid client for this security server.

[SEC_AUTHORIZATION_TARGETID_INVALID]
The target function or object for which authorization is being requested is invalid.

[SEC_AUTHORIZATION_TARGETPERM_INVALID]
The requested permission is not defined for the target string. This may indicate that the SRF and the security server are out of sync.

Examples

A typical call to this function follows:

```
  #include <sec_api.h>
  #include <sec_errs.h>

main(){
          int status, authorized;

/***
* Establishes User's security context.
* If denied, must exit.
***/
nvs_isClientAuthorized ("nmpolling", "x", &authorized, &status);
if ((status != SEC_SUCCESS) || (! authorized))
   {
   fprintf(stderr, "><nvs_isClientAuthorized(): permission denied for
\"%s\".\n", "nmpolling");
   fprintf(stderr, "< %s authorized=%d, status=%d.\n",
nvs_SecErrMsg(status), authorized, status );

     nvs_deleteSecContext ();
     exit(1);
```

```
    }
}
```

Libraries

When compiling a program that uses nvs_isClientAuthorized, link to the following shared libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_isClientAuthorized, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_Audit(3)” on page 384.
- See “nvs_deleteSecContext(3)” on page 386.
- See “nvs_getClientPerms(3)” on page 388.

nvs_SecErrMsg(3)

Purpose

Returns status message from security API calls

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/* This second include statement applies only if you want to parse error codes */

char * nvs_SecErrMsg ( int status)
```

Description

This function returns a message about the status of an API call based on the status value returned on the call. This routine can be used with nvs_getClientPerms and nvs_isClientAuthorized calls to return a text string indicating the success or failure of these calls.

Parameters

status	The status of the call, in a human-readable format.

Libraries

When compiling a program that uses nvs_SecErrMsg, link to the following shared libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_SecErrMsg, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_Audit(3)” on page 384.
- See “nvs_deleteSecContext(3)” on page 386.
- See “nvs_getClientPerms(3)” on page 388.

nvs_isSecOn(3)

Purpose

Determines whether NetView for AIX security is active.

Syntax

```
#include <sec_api.h>
#include <sec_errs.h>
/* This second include statement applies only if you want to parse error codes */

int nvs_isSecOn ( );
```

Description

This function checks to see if NetView for AIX security is active. This function should be called after a call to nvs_isClientAuthorized to check a user's execute (x) authorization for a function.

Return Values

If security is active, nvs_isSecOn returns NVSEC_ON. If security is not active, nvs_isSecOn returns NVSEC_OFF.

Libraries

When compiling a program that uses nvs_isSecOn, link to the following shared libraries:

- /usr/OV/lib/libnvsec.a
- /usr/OV/lib/libnvgss.a

Files

When compiling a program that uses nvs_isSecOn, you need to include the following files:

- sec_api.h
- sec_errs.h

Related Information

- See “nvs_Audit(3)” on page 384.
- See “nvs_deleteSecContext(3)” on page 386.
- See “nvs_getClientPerms(3)” on page 388.

om_copy(3)

Purpose

Duplicates a private object

Syntax

```
#include  <xom.h>

OM_return_code om_copy( OM_private_object original,
                        OM_workspace workspace,
                        OM_private_object *copy );
```

Description

The om_copy routine creates a new private OM object, the copy, that is an exact, but independent, copy of an existing private object, the original. The function is recursive in that copying the original also copies its subobjects.

Parameters

original	Specifies the private OM object to be copied. This OM object remains accessible.
workspace	Specifies the workspace in which the copy is to be created. The original's class must be in a package associated with this workspace.
copy	Specifies the result. It is present if, and only if, the function result is success.

Return Values

If successful, om_copy returns [OM_SUCCESS]. If unsuccessful, om_copy returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED]	The function does not apply to the object to which it is addressed.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a key-stroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_CLASS]	A purported class identifier is undefined.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_WORKSPACE]	A purported workspace is undefined.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.

[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_TOO_MANY_VALUES]	An implementation limit prevents the addition to an object of another attribute value. This limit is undefined.

Files

When compiling a program that uses om_copy, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_copy, you need to link to the following library:

/usr/OV/lib/libxmp.a

om_copy_value(3)

Purpose

Copies a string value from one private object to another

Syntax

```
#include  <xom.h>

OM_return_code om_copy_value( OM_private_object source,
                              OM_type source_type,
                              OM_value_position source_value_position,
                              OM_private_object destination,
                              OM_type destination_type,
                              OM_value_position destination_value_position );
```

Description

This function copies a string value from one private OM object, the source, to another private OM object, the destination. This function can be used to add a new value to the destination or to replace an existing value. The syntax of the source value must be one of the string types. The copy's syntax is that of the original.

Parameters

source	Specifies the source OM object, which remains accessible. This must be a private OM object.
source_type	Specifies the type of the attribute that has a value to be copied.
source_value_position	Specifies the position within the above attribute of the value to be copied. When an OM Class Definition specifies 0 (zero)-or-more or 1-or-more in the Value Number field, this parameter specifies which of the values is to be used. Each of the values for an attribute is represented by an OM Descriptor numbered from 0 to n-1. A value of zero is used when only one value is allowed for this attribute type, or to specify the first OM descriptor of this type when multiples are allowed.
destination	Specifies the destination OM object, which remains accessible. This must be a private OM object.
destination_type	Specifies the type of the attribute to which the value should be copied.
destination_value_position	Specifies the position within the above attribute of the value to be added or replaced. If the value position exceeds the number of values present in the destination attribute, the argument is taken to be equal to that number. This allows the new value to be added after all existing values. If the value position is less than the number of values currently in the destination, this parameter identifies which value is to be replaced.

Return Values

If successful, om_copy_value returns [OM_SUCCESS]. If unsuccessful, om_copy_value returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED]	The function does not apply to the object to which it is addressed.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a key-stroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_PRESENT]	An attribute value is absent, not present.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_WRONG_VALUE_LENGTH]	An attribute has, or would have, a value that violates the value length constraints in force.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.
[OM_WRONG_VALUE_TYPE]	An object has, or would have, an attribute whose type is not permitted.

Files

When compiling a program that uses om_copy_value, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_copy_value, you need to link to the following library:

/usr/OV/lib/libxmp.a

om_create(3)

Purpose

Creates a new private object

Syntax

```
#include  <xom.h>

OM_return_code om_create( OM_object_identifier    class,
                          OM_boolean              initialize,
                          OM_workspace            workspace,
                          OM_private_object       *object );
```

Description

This function creates a new private object that is an instance of a particular OM class.

Parameters

class Specifies the OM class of the OM object to be created. The specified OM class must be concrete.

initialize Specifies whether the created object is to be initialized as specified in the definition of its OM class. If this argument is true, the OM object will contain all attributes that have initial values defined in the OM Class Definition for this class and all of its super classes. If this argument is false, the OM object will contain only the Class attribute.

workspace
Specifies the workspace in which the OM object is to be created. The specified class must be in a package associated with this workspace.

object Specifies the newly created OM object. This result is present if, and only if, the function result is success.

Return Values

If successful, om_create returns [OM_SUCCESS]. If unsuccessful, om_create returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED]	The function does not apply to the object to which it is addressed.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_CLASS]	A purported class identifier is undefined.
[OM_NO_SUCH_WORKSPACE]	A purported workspace identifier is undefined.

[OM_NOT_CONCRETE]	A class is abstract, not concrete.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.

Files

When compiling a program that uses om_create, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_create, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_delete(3)” on page 404.

om_decode(3)

Purpose

Creates a unencoded version of an encoded private object

Syntax

```
#include  <xom.h>

OM_return_code om_decode( OM_private_object encoding,
                          OM_private_object *original );
```

Description

This function creates a new, private OM object. This object, called the original, is an exact but independent copy of the OM object encoded by an existing private OM object, called the encoding.

Parameters

encoding Specifies the encoding, which remains accessible. It shall be an instance of OM class Encoding.

original Specifies the original, which is created in the encoding's workspace. This result is present if, and only if, the function result is success.

Return Values

If successful, om_decode returns [OM_SUCCESS]. If unsuccessful, om_decode returns one of the following error codes.

Error Codes

[OM_ENCODING_INVALID]	The octets that constitute the value of an encoding's Object Encoding attribute are not valid.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_CLASS]	A purported class identifier is undefined.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_RULES]	A purported rules identifier is undefined.
[OM_NOT_AN_ENCODING]	An object is not an instance of the Encoding class.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.

[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_TOO_MANY_VALUES]	An implementation limit prevents the addition to an object of another attribute value. This limit is undefined.
[OM_WRONG_VALUE_LENGTH]	An attribute has, or would have, a value that violates the value length constraints in force.
[OM_WRONG_VALUE_MAKEUP]	An attribute has, or would have, a value that violates a constraint of the value's syntax.
[OM_WRONG_VALUE_NUMBER]	An attribute has, or would have, a value that violates the value number constraints in force.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.
[OM_WRONG_VALUE_TYPE]	An object has, or would have, an attribute whose type is not permitted.

Files

When compiling a program that uses om_decode, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_decode, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_encode(3)” on page 406.

om_delete(3)

Purpose

Deletes a private or service-generated object

Syntax

```
#include  <xom.h>

OM_return_code om_delete( OM_object subject );
```

Description

This function deletes a service-generated public OM object or a private OM object. It is not intended for use on client-generated public OM objects.

If applied to a service-generated public OM object, the function deletes the OM object and releases any resources associated with the OM object, including the space occupied by descriptors and attribute values. The function is applied recursively to any public subobjects. There is no effect on private subobjects. You should not use om_delete directly on public subobjects, which are subobjects existing within the public OM object copy returned from a call to *om_get* with no exclusions. You should apply the function only to the top-level OM object, which is the object pointed to by copy. Otherwise you might get unspecified results.

If applied to a private object, the function makes the OM object inaccessible. Existing OM object handles for the OM object are determined not valid. The function is applied recursively to any private subobjects.

Parameters

subject Specifies the OM object that is to be deleted.

Return Values

If successful, om_delete returns [OM_SUCCESS]. If unsuccessful, om_delete returns one of the following error codes.

Error Codes

[OM_MEMORY_INSUFFICIENT]
The service cannot allocate the main memory it needs to complete the function.

[OM_NETWORK_ERROR] The service could not successfully use the network upon which its implementation depends.

[OM_NO_SUCH_OBJECT] A purported object is nonexistent or the purported handle is not valid.

[OM_NO_SUCH_SYNTAX] A purported syntax identifier is undefined.

[OM_NO_SUCH_TYPE] A purported type identifier is undefined.

[OM_NOT_THE_SERVICES] An object is client-generated, rather than service-generated or private.

[OM_PERMANENT_ERROR] The service encountered a permanent difficulty, other than those denoted by other return codes.

[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.

Files

When compiling a program that uses om_delete, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_delete, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_create(3)” on page 400.
- See “om_get(3)” on page 408.

om_encode(3)

Purpose

Encodes an OM object

Syntax

```
#include  <xom.h>

OM_return_code om_encode( OM_private_object original,
                          OM_object_identifier rules,
                          OM_private_object *encoding );
```

Description

This function creates a new, private OM object, called the encoding, that exactly and independently encodes an existing private OM object, called the original. You can identify the set of rules that this function follows to produce the encoding.

The definition of a package identifies zero or more of its concrete classes to which this function applies.

Parameters

original Specifies the original OM object to be encoded. This OM object remains accessible.

rules Specifies the set of rules that the function follows to produce the encoding. The defined values of this argument are those of the Rules attribute specific to the Encoding class. The identifier which should be used for this parameter is OM_BER as defined in the xom.h include file.

encoding Specifies the resulting encoded OM object, an instance of class Encoding, which is created in the original's workspace. This result is present if, and only if, the function result is success.

Return Values

If successful, om_encode returns [OM_SUCCESS]. If unsuccessful, om_encode returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED] The function does not apply to the object to which it is addressed.

[OM_FUNCTION_INTERRUPTED] The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.

[OM_MEMORY_INSUFFICIENT] The service cannot allocate the main memory it needs to complete the function.

[OM_NETWORK_ERROR] The service could not successfully use the network upon which its implementation depends.

[OM_NO_SUCH_OBJECT] A purported object is nonexistent or the purported handle is not valid.

[OM_NO_SUCH_RULES]	A purported rules identifier is undefined.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.

Files

When compiling a program that uses om_encode, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_encode, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_decode(3)” on page 402.

om_get(3)

Purpose

Creates a public copy of all or particular parts of a private object

Syntax

```
#include  <xom.h>

OM_return_code om_get( OM_private_object original,
                       OM_exclusions exclusions,
                       OM_type_list included_types,
                       OM_boolean local_strings,
                       OM_value_position initial_value,
                       OM_value_position limiting_value,
                       OM_public_object *copy,
                       OM_value_position *total_number );
```

Description

This function creates a new public OM object that is an exact, but independent, copy of an existing private OM object. This provides the ability to retrieve attribute values from a private OM object or to determine the structure of the private OM object. Certain exclusions might be requested, which reduces the copy to a portion of the original.

One exclusion is always requested implicitly. For each attribute value in the original that is a string whose length exceeds an implementation-defined number, the copy includes a descriptor that omits the elements, but not the length, of the string. The elements component of the string component of the value component of the descriptor is elements-unspecified, and the Long-String bit of the syntax component is set to true.

You can access long values by using the om_read function.

Parameters

original Specifies the OM object to be copied. This OM object remains accessible.

exclusions

Specifies zero or more exclusions, each of which reduces the copy to a prescribed portion of the original. The exclusions apply to the attributes of the OM object but not to those of its subobjects. This allows the user to only retrieve the portions of the private OM object which are of interest.

Each value except no-exclusions is chosen from the following list. When multiple exclusions are specified, each is applied in the order in which it appears in the list with lower-numbered exclusions having precedence over higher-numbered exclusions. If, after the application of an exclusion, that portion of the OM object would not be returned, no further exclusions need be applied to that portion.

The following list describes the possible types of exclusions:

(1) exclude-all-but-these-types — The copy includes descriptors that encompass only attributes of specified types. This type of exclusion enables your to define the attributes of the private OM object that are of interest.

(2) exclude-multiples	The copy includes a single descriptor for each attribute having two-or-more values, rather than one descriptor for each value. Each such descriptor contains no attribute value and the No-Value bit of the syntax component is set. If the attribute has values of two-or-more syntaxes, the descriptor identifies one of those syntaxes but does not specify which one. This exclusion determines the presence of multivalued attributes without simultaneously getting their values.
(3) exclude-all-but-these-values	The copy includes descriptors that encompass only values at specified positions within an attribute. When used in conjunction with the exclude-all-but-these-types exclusion, this exclusion determines the values of a specified attribute, as well as the syntaxes of those values using one or more attributes, but not all attributes, at a time. This functionality enables processing of subsets of a multivalued attribute instead of processing all values of an attribute at a time.
(4) exclude-values	The copy includes a single descriptor for each attribute value, but the descriptor does not contain the value, and the No-Value bit of the syntax component is set. This exclusion determines an object's composition, that is, the type and syntax of each of its attribute values.
(5) exclude-subobjects	The copy includes, for each value whose syntax is object, a descriptor containing an object handle for the original private subobject, rather than a public copy of it. This handle makes that subobject accessible for use in subsequent function calls. This exclusion examines an object one level at a time.
(6) exclude-descriptors	No descriptors are returned and the copy result is not present. The total_number result reflects the number of descriptors that would have been returned by applying the other inclusion and exclusion specifications. This exclusion provides an attribute analysis capability. For example, the total number of values in a multivalued attribute can be determined by specifying an inclusion of the specific attribute type and the following exclusions: exclude-all-but-these-types, exclude-subobjects, and exclude-descriptors. The exclude-all-but-these-values exclusion affects the choice of descriptors and the exclude-values exclusion affects the composition of descriptors.

included_types

Is present if, and only if, the exclude-all-but-these-types exclusion is requested. This is an array listing the types to be included, and must end with the NULL terminator OM_NO_MORE_TYPES.

local_strings
If true, indicates that all string(*) values included in the copy are to be translated into the implementation-defined local character set representation, which can cause some information to be lost. This feature is not supported in the current implementation.

initial_value
Is present if, and only if, the exclude-all-but-these-values exclusion is requested. This field specifies that the position within each attribute of the first value is included in the copy. If this field is all-values or exceeds the number of values present in an attribute, the argument is equal to that number.

limiting_value
Specifies the position within each attribute one beyond that of the last value to be included in the copy. It is present if, and only if, the exclude-all-but-these-values exclusion is requested. If this argument is not greater than initial_value, no values are included and no descriptors are returned. If this field is all-values or exceeds the number of values present in an attribute, the argument is equal to that number.

copy Specifies the copy. This result is present if, and only if, the function result is success and the exclude-descriptors exclusion is not specified. The space occupied by the public OM object and every attribute value that is a string is service-provided. If you alter any portion of that space, the effect on the service's subsequent behavior is unspecified. The space occupied by this OM object should be freed when no longer needed with om_delete.

total_number
Specifies the number of attribute descriptors returned in the public OM object, but not in any of its subobjects, based on the inclusion and exclusion arguments specified. This result is present if, and only if, the function result is success. If the exclude-descriptors exclusion is specified, no copy result is returned and the total_number result reflects the actual number of attribute descriptors that would have been returned, based on the remaining inclusion and exclusion values. The total includes only the attribute descriptors in the copy result, including the first descriptor which specifies the class of the public OM object. It excludes the special descriptor signalling the end of the public OM object.

Return Values

If successful, om_get returns [OM_SUCCESS]. If unsuccessful, om_get returns one of the following error codes.

Error Codes

[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a key-stroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_EXCLUSION]	A purported exclusion identifier is undefined.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_PRIVATE]	An object is public, not private.

[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.
[OM_WRONG_VALUE_TYPE]	An object has, or would have, an attribute whose type is not permitted.

Files

When compiling a program that uses om_get, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_get, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_delete(3)” on page 404.
- See “om_put(3)” on page 414.

om_instance(3)

Purpose

Checks the class of an object

Syntax

```
#include  <xom.h>

OM_return_code om_instance( OM_object subject,
                            OM_object_identifier class,
                            OM_boolean *instance );
```

Description

This function determines whether a service-generated public or private OM object, the subject, is an instance of a particular class or any of its subclasses.

You can determine an object's class, C, by inspecting the object, using programming-language constructions if the object is public or the om_get function if it is private. The om_instance function shows that an object is an instance of the specified class, even if C is a subclass of that class.

Parameters

subject	Specifies the OM object which is to checked. This OM object remains accessible.
class	Specifies the class in question.
instance	Specifies a boolean that identifies whether the subject is an instance of the specified class or any of its subclasses. This result is present if, and only if, the function result is success.

Return Values

If successful, om_instance returns [OM_SUCCESS]. If unsuccessful, om_instance returns one of the following error codes.

Error Codes

[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_CLASS]	A purported class identifier is undefined.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_SYNTAX]	A purported syntax identifier is undefined.

[OM_NOT_THE_SERVICES]	An object is client-generated, rather than service-generated or private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.

Files

When compiling a program that uses om_instance, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_instance, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_get(3)” on page 408.

om_put(3)

Purpose

Adds or replaces attributes in a private object

Syntax

```
#include  <xom.h>

OM_return_code om_put( OM_private_object destination,
                       OM_modification modification,
                       OM_object source,
                       OM_type_list included_types,
                       OM_value_position initial_value,
                       OM_value_position limiting_value );
```

Description

This function copies attributes from an OM object into a private OM object. These attributes can be additions to the attributes in the destination OM object, or they might replace existing attributes in the destination OM object. You can specify that the source's values are to replace all values or particular values in the destination or to be inserted at a particular position within each attribute.

This function does not insert the source OM object into the destination; it copies attributes from the source to the destination. The class of the source OM object does not have to match the class of the destination OM object. Subject to the modifications and included_types, this function traverses the source object and copies all attributes that are valid for the destination object into the destination OM object. Attributes that exist in the source OM object but do not exist in the destination OM object's class are ignored.

Parameters

destination

Specifies the OM object to be modified. This OM object remains accessible and its class is unaffected.

modification

Specifies the nature of the requested modification. The modification field determines how this function uses the attribute values in the source to modify the object. In all cases, for each attribute present in the source, copies of its values are placed in the object's destination attribute of the same type. The data value is chosen from among the following values:

(1) insert-at-beginning	The source values are inserted before any existing destination values These destination values are retained.
(2) insert-at-certain-point	The source values are inserted before the value at a specified position in the destination attribute. If this value is used, the inital_value field is used to indicate the specified position.
(3) insert-at-end	The source values are inserted after any existing destination values.
(4) replace-all	The source values are placed in the destination attribute. The existing destination values, if any, are discarded.

(5) replace-certain-values — The source values are substituted for the values at specified positions in the destination attribute, the latter are discarded. If this value is used, the inital_value field is used to indicate the starting position and limiting_value is used to indicate the ending position of the attribute values to be replaced.

source Specifies the source OM object, which remains accessible. The source's class is ignored. However, the attributes being copied from the source must be compatible with the destination's class definition.

included_types
If present, specifies the types of the attributes to be included in the destination (provided that they appear in the source); otherwise, all attributes are to be included. This is an array of types and must end with the NULL terminator OM_NO_MORE_TYPES.

initial_value
Is present if, and only if, the modification argument is insert-at-certain-point or replace-certain-values, the position within each destination attribute at which source values are to be inserted, or of the first value to be replaced, respectively. If it is all-values or exceeds the number of values present in a destination attribute, the argument is equal to that number. The values of an attribute are numbered from 0 to n–1.

limiting_value
Is present if, and only if, the modification argument is replace-certain-values, the position within each destination attribute, one beyond that of the last value to be replaced. If this argument is present, it must be greater than the initial_value argument. If it is all-values or exceeds the number of values present in a destination attribute, the argument is equal to that number.

Return Values

If successful, om_put returns [OM_SUCCESS]. If unsuccessful, om_put returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED]	The function does not apply to the object to which it is addressed.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory that it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_CLASS]	A purported class identifier is undefined.
[OM_NO_SUCH_MODIFICATION]	A purported modification identifier is undefined.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_SYNTAX]	A purported syntax identifier is undefined.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_CONCRETE]	A class is abstract, not concrete.
[OM_NOT_PRESENT]	An attribute value is absent, not present.

[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_TOO_MANY_VALUES]	An implementation limit prevents the addition to an object of another attribute value. This limit is undefined.
[OM_VALUES_NOT_ADJACENT]	The descriptors for the values of a particular attribute are not adjacent.
[OM_WRONG_VALUE_LENGTH]	An attribute has, or would have, a value that violates the value length constraints in force.
[OM_WRONG_VALUE_MAKEUP]	An attribute has, or would have, a value that violates a constraint of the value's syntax.
[OM_WRONG_VALUE_NUMBER]	An attribute has, or would have, a value that violates the value number constraints in force.
[OM_WRONG_VALUE_POSITION]	A value position identified in the argument of a function is not valid.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.
[OM_WRONG_VALUE_TYPE]	An object has, or would have, an attribute whose type is not permitted.

Files

When compiling a program that uses om_put, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_put, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_delete(3)” on page 404.
- See “om_get(3)” on page 408.

om_read(3)

Purpose

Reads a string segment in a private object

Syntax

```
#include  <xom.h>

OM_return_code om_read( OM_private_object subject,
                        OM_type type,
                        OM_value_position value_position,
                        OM_boolean local_string,
                        OM_string_length *string_offset,
                        OM_string *elements );
```

Description

This function reads a segment of an attribute value in a private OM object, the subject. The segment that is returned is a segment of the string value that would have been returned if the complete value had been read in a single call.

Note that this function enables the client to read an arbitrarily long value without requiring that the service place a copy of the entire value in memory.

Parameters

subject	Specifies the OM object to be read. This OM object remains accessible.
type	Specifies the type of the attribute, one of whose values is to be read.
value_position	Specifies the position within the above attribute of the value to be read. When the attribute allows multiple values (multiple OM descriptors with the same attribute type), this value indicates which value to return. The values are numbered from 0 to n–1.
local_string	If true, indicates that the value is to be translated into the implementation-defined local character set representation. This feature is not supported.
string_offset	On input, this value contains the offset, in octets, of the start of the string segment to be read. If it exceeds the total length of the string, the argument is taken to be equal to the string length. Upon successful return from this call, this value will contain the offset, in octets, of the start of the next string segment to be read, or zero if the value's final segment was read.
elements	On input, this value contains a String structure that contains the number of octets to be read. The pointer value in the String structure should be set to point to a user-supplied buffer where the string can be stored. Upon successful return from this call, this value will contain the offset, in octets, of the start of the next string segment to be read, or zero if the value's final segment was read.

Return Values

If successful, om_read returns [OM_SUCCESS]. If unsuccessful, om_read returns one of the following error codes.

Error Codes

[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory that it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_PRESENT]	An attribute value is absent, not present.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.

Files

When compiling a program that uses om_read, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_read, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_write(3)” on page 421.

om_remove(3)

Purpose

Removes attribute values from a private object

Syntax

```
#include  <xom.h>

OM_return_code om_remove( OM_private_object subject,
                          OM_type type,
                          OM_value_position initial_value,
                          OM_value_position limiting_value );
```

Description

This function removes and discards particular values of an attribute of a private OM object, the subject. If no values remain, the attribute is also removed. If the value is a subobject, the value is first removed and then the om_delete function is applied to it, deleting the object.

Parameters

subject Specifies the OM object, which remains accessible. The subject's class is unaffected.

type Specifies the type of the attribute, which has some values to be removed. The type should not be Class.

initial_value Specifies the position within the above attribute of the first value to be removed. If it is all-values or exceeds the number of values present in the attribute, the argument is equal to that number. In a multivalued attribute (multiple OM descriptors with the same attribute type), this value indicated which descriptor to start with. The values are numbered from zero to n–1.

limiting_value Specifies the position within the attribute one beyond that of the last value to be removed. If this argument is not greater than the initial_value argument, no values are removed. If it is all-values or exceeds the number of values present in an attribute, the argument is equal to that number.

Return Values

If successful, om_remove returns [OM_SUCCESS]. If unsuccessful, om_remove returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED] The function does not apply to the object to which it is addressed.

[OM_FUNCTION_INTERRUPTED] The function was stopped by an external force, such as a keystroke, that was designated for this purpose in a user interface.

[OM_MEMORY_INSUFFICIENT] The service cannot allocate the main memory that it needs to complete the function.

[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.

Files

When compiling a program that uses om_remove, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_remove, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_delete(3)” on page 404.
- See “om_put(3)” on page 414.

om_write(3)

Purpose

Writes a segment of a string into a private object

Syntax

```
#include  <xom.h>

OM_return_code om_write( OM_private_object subject,
                         OM_type type,
                         OM_value_position value_position,
                         OM_syntax syntax,
                         OM_string_length *string_offset,
                         OM_string elements );
```

Description

This function writes a segment of an attribute value in a private OM object, the subject. The segment that is supplied is a segment of the string value that would have been supplied if the complete value had been written in a single call. The written segment is made the value's last; the function discards values whose offset equals or exceeds the string_offset argument.

This function enables you to write an arbitrarily long value without having to place a copy of the entire value in memory.

Parameters

subject Specifies the OM object to be written. This OM object remains accessible.

type Specifies the type of the attribute, one of whose values is to be written.

value_position Specifies the position within the above attribute of the value to be written. The value position shall neither be negative nor exceed the number of values present. If it equals the number of values present, the segment is inserted into the attribute as a new value. In a multivalued attribute (multiple OM descriptors are allowed with the same attribute type), this parameter identifies the descriptor to modify. The descriptors are numbered from zero to n–1.

syntax Specifies the syntax the value is to have if the value being written was not already present in the subject. It must be a permissible syntax for the attribute of which this is a value. If the value being written was already present in the subject, that value's syntax is preserved and this argument is ignored.

string_offset On input, this value contains the offset, in octets, of the start of the string segment to write. If it exceeds the current length of the string value being written, the argument is taken to be equal to that current length. Upon successful return from this call, this value will contain the offset, in octets, of the start of the next string segment to be written. This values enables the value of a long string to be written sequentially.

elements Specifies the string segment to be written. A copy of this segment will occupy a position within the string value being written, starting at the offset given by the string_offset argument. Any values located at or beyond this offset are discarded.

Return Values

If successful, om_write returns [OM_SUCCESS]. If unsuccessful, om_write returns one of the following error codes.

Error Codes

[OM_FUNCTION_DECLINED]	The function does not apply to the object to which it is addressed.
[OM_FUNCTION_INTERRUPTED]	The function was stopped by an external force, such as a key-stroke, that was designated for this purpose in a user interface.
[OM_MEMORY_INSUFFICIENT]	The service cannot allocate the main memory that it needs to complete the function.
[OM_NETWORK_ERROR]	The service could not successfully use the network upon which its implementation depends.
[OM_NO_SUCH_OBJECT]	A purported object is nonexistent or the purported handle is not valid.
[OM_NO_SUCH_SYNTAX]	A purported syntax identifier is undefined.
[OM_NO_SUCH_TYPE]	A purported type identifier is undefined.
[OM_NOT_PRESENT]	An attribute value is absent, not present.
[OM_NOT_PRIVATE]	An object is public, not private.
[OM_PERMANENT_ERROR]	The service encountered a permanent difficulty, other than those denoted by other return codes.
[OM_POINTER_INVALID]	A pointer that is not valid was supplied as a function argument or as the receptacle for a function result.
[OM_SYSTEM_ERROR]	The service could not successfully use the operating system upon which its implementation depends.
[OM_TEMPORARY_ERROR]	The service encountered a temporary difficulty, other than those denoted by other return codes.
[OM_WRONG_VALUE_LENGTH]	An attribute has, or would have, a value that conflicts with the value length rules in effect.
[OM_WRONG_VALUE_MAKEUP]	An attribute has, or would have, a value that conflicts with a rule of the value's syntax.
[OM_WRONG_VALUE_POSITION]	A value position identified in the argument of a function is not valid.
[OM_WRONG_VALUE_SYNTAX]	An attribute has, or would have, a value whose syntax is not permitted.

Files

When compiling a program that uses om_write, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses om_write, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “om_read(3)” on page 417.

OVDefaultServerName(3)

Purpose

Determines the name of the default server to which a client should connect

Syntax

```
#include  <nvDefServ.h>

char * OVDefaultServerName ()
```

Description

This function determines the name of the server that a client application should connect to in a client/server environment. The application returns the hostname of the machine to which the client will be connecting.

The default server name is stored in a file called /usr/OV/databases/servername. This file exists only on clients. If the file exists on the target machine, OVDefaultServerName returns the contents of the file. If the file does not exist, the machine is a server, and OVDefaultServerName returns the current hostname.

Return Values

If successful, OVDefaultServerName returns a server name.

Files

When compiling a program that uses OVDefaultServerName, you need to include the following file:

- nvDefServ.h

Libraries

When compiling a program that uses OVDefaultServerName, link to the following library:

/usr/OV/lib/libOVW.a

Related Information

- See “NVisClient(3)” on page 136.

OVeDeregister(3)

Purpose

Deregisters the caller from receiving events from the listed network nodes

Syntax

```
#include <xom.h>
#include <xmp.h>
#include <xmp_cmis.h>
#include <ove_xmp.h>

int OVeDeregister ( OM_private_object session,
                    OM_workspace workspace,
                    OVeConvDeRegNode *sieve_list,
                    OVeConvConfirm **confirm_list,
                    OM_return_code *om_error);
```

Description

When an application or agent deregisters for events, it longer receives a set of events for which it was previously registered. The caller identifies the sieve objects and locations returned in the confirmation response to the OVeRegister call, and calls OVeDeregister. This list should include the local event sieve, as well as any remote sieve objects, unless the caller is deleting only a subset of remote sieves created.

Deregistering is a confirmed CMIS action; confirmation of the deletion or an error is received from EMS. OVeDeregister returns the invoke IDs of the requests so they can be matched with the event sieve confirmations returned from EMS. The confirmation responses must be received with mp_receive.

The scoping and filtering of object instances is not supported. The OVeDeregister routine accepts a list of nodes, by hostname or address, and corresponding event sieve object instances and deletes them. Use of pattern-matching characters is not supported.

Parameters

session
Specifies an OM_private_object (returned from mp_bind) that identifies the binding established with the Communications Infrastructure.

workspace
Specifies the XOM workspace used in building the delete requests.

sieve_list
Specifies a pointer to a linked list of destination addresses, object classes, and instances returned by the OVeRegister confirmations that define the event sieve objects to be deleted. The destination address can be either a host name or address. The object class/instance is provided in the OM_object structure that was previously returned by mp_receive.

confirm_list
Specifies a pointer to a linked list of OVeConvConfirm structures. The structures contain information for each delete request that OVeDeregister made: the destination node address, the invoke ID returned from the mp_delete_req, and the status returned from the mp_delete_req. The caller needs to receive, through mp_receive, a confirmation of each delete request that was made in order to ensure that each event sieve was successfully deleted.

om_error
Specifies a pointer to an OM_return_code returns if XOM errors occurred in sending the delete requests. This pointer is valid only if the return value indicates an XOM error has occurred.

Return Values

If successful, OVeDeregister returns 0 (zero). If unsuccessful, it returns one of the negative integers listed in the ove_xmp.h header file.

Files

When compiling a program that uses OVeDeregister, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses OVeDeregister, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “OVeRegister(3)” on page 431.
- See “mp_receive(3)” on page 76.
- See “mp_delete_req(3)” on page 55.

OVeFilterAttr(3)

Purpose

Builds event filter structures

Syntax

```
#include <xom.h>
#include <xmp.h>
#include <xmp_cmis.h>
#include <OV/ove_xmp.h>

int OVeFilterAttr ( OM_workspace workspace,
                    char *filter_string,
                    OM_private_object *out_filter_attribute,
                    char **err_ptr,
                    OM_return_code *om_error);
```

Description

OVeFilterAttr generates an event filter attribute that may subsequently be used in a call to OVeRegister. OVeFilterAttr accepts a string defining a filter, and parse it into the appropriate structure.

Parameters

workspace
Specifies the XOM workspace in which the returned XOM object will be created.

filter_string
Specifies a pointer to a NULL-terminated string that defines the filter using key words and values. The string is interpreted left-to-right, unless otherwise indicated by parentheses. The syntax for the string is described under Filter String Syntax on page 427.

out_filter_attribute
Specifies a pointer returning an OM_private_object structure defining the event filter attribute that can be subsequently passed to the OVeRegister convenience routine. The object is an instance of the OM class Attribute.

err_ptr
Specifies a pointer to the input string, indicating where the string parser detected errors. This pointer is valid only if an error occurred and can point to the whole string depending on the context of the error.

om_error
Specifies pointer to an OM_return_code returned if any XOM errors occurred in processing the event filter.

Filter String Syntax: The filter_string is a NULL-terminated string parsed by OVeFilterAttr to create the appropriate XOM object for a filter. Keywords are provided to determine the type of filtering, with comparison values attached using the standard comparison operator <=, >=, or =. Keywords must be separated from preceding tokens by white space, except when they are preceded by a left parenthesis. In this case, the left parenthesis must be separated by white space. Nested parentheses are supported, and should be used for logical groupings. Otherwise, precedence is as read from left to right.

Keyword Syntax

! NOT (logical negation)

&& AND (logical and)

|| OR (logical or)

The following list describes the keyword in the syntax used to define filters.

CLASS=value
Object class match on class ID. Value is given in dot notation, for example, 1.2.3.4.55

CLASS=value
SNMP enterprise match on enterprise ID. Value is given in dot notation, for example, 1.2.3.4.55

IP_ADDR=value
Object instance match on IP address. Value is given in dot notation, for example, 192.155.13.57

IP_ADDR=value
SNMP agent-addr match on IP address. Value is given in dot notation, for example, 192.155.13.57 Registration for an IP_ADDR permits receipt of agent-generated traps as well as internal events related to that IP_ADDR.

Note: IP_ADDR permits an application to register to receive NetView for AIX internal events related to IP_ADDR as well as traps originating from IP_ADDR.

FDN=value
Object instance match on Fully Distinguished Name See the following syntax for FDN.

EVENT_TYPE=value
Match on Event type id. Value is given in dot notation, for example, 1.2.3.4.55.

PRESENT=EVENT_TIME
Presence of Event Time

EVENT_TIME <= time_string
Event time before value in time_string, where time_string is of the form dd:mm:yy:hh:mm:ss (24-hour clock, GMT)

EVENT_TIME >= time_string
Event time after value in time_string, where time_string is of the form dd:mm:yy:hh:mm:ss (24-hour clock, GMT)

LOGGED_TIME <= time_string
Logged time before value in time_string, where time_string is of the form dd:mm:yy:hh:mm:ss (24-hour clock, GMT)

LOGGED_TIME >= time_string
Logged time after value in time_string, where time_string is of the form dd:mm:yy:hh:mm:ss (24-hour clock, GMT)

PRESENT = SNMP_TRAP
Presence of SNMP Trap

SNMP_TRAP=value
Match on SNMP Generic Trap Type, where the Generic Type is an integer

SNMP_SPECIFIC=value
Match on SNMP Specific Trap Type, where the Specific Type is an integer

TIME_PERIOD=time_constant
Relative time period (integer seconds) for frequency filters

THRESHOLD <= frequency
Number of event occurrences is less than or equal to frequency (integer) during TIME_PERIOD

THRESHOLD >= frequency
Number of event occurrences is greater than or equal to frequency (integer) during TIME_PERIOD

Note: When included in an expression for OVeFilterAttr, the keywords THRESHOLD and TIME_PERIOD must be ANDed, never ORed, and grouped within parentheses as shown in the following example:

```
filter = PRESENT=SNMP_TRAP && (THRESHOLD <= 5 && TIME_PERIOD = 30)
```

Specifying more than 250 filter objects will result in an error in the OVeRegister call to register the filter.

FDN Syntax: For object instance filtering, OVeFilterAttr supports specification of an FDN (Fully Distinguished Name) with the most common value set. The filter string value for an FDN has the following syntax:

```
FDN :: =/RDN [/RDN] ...

RDN :: =attribute = value [; attribute = value] ...

attribute :: = object_instance_attribute_id in dot notation

value ::= INT integer | STRING "string" | ADDRESS CMOT_system_id |
           TIME dd:mm:yy:hh:mm:ss | OID object_id
```

An ADDRESS must be an IP Address specified as a CMOT System ID with an attribute ID of 1.3.6.1.2.1.9.3.

Examples

```
FDN = /1.2.3.4.5.6 = INT 69 /1.3.5=STRING "abcd"

FDN = /1.3.6.1.2.1.9.3 = ADDRESS 1.2.3.4

PRESENT = EVENT_TIME || IP_ADDR = 192.6.6.6

(EVENT_TIME <= 09:10:90:08:00:00)  &&  (EVENT_TIME>= 09:10:90:17:00:00)

(EVENT_TYPE = 1.3.6.1.4.1.11.2.2.6.9)  ||  (CLASS  = 1.3.6.1.4.1.11.2.0.0)

SNMP_TRAP = 4  &&  (TIME_PERIOD = 60  &&  THRESHOLD > = 10)
```

Return Values

If successful, OVeFilterAttr returns 0 (zero). If unsuccessful, it returns a negative integer listed in the ove_xmp.h header file.

Files

When compiling a program that uses OVeFilterAttr, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses OVeFilterAttr, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “OVeRegister(3)” on page 431.
- See “mp_get_req(3)” on page 69.

OVeRegister(3)

Purpose

Registers the caller with EMS to receive filtered events from the listed network nodes

Syntax

```
#include <xom.h>
#include <xmp.h>
#include <xmp_cmis.h>
#include <ove_xmp.h>

int OVeRegister ( OM_private_object session,
                  OM_workspace workspace,
                  OVeConvRegNode *node_list,
                  OM_private_object filter_attribute,
                  OVeConvConfirm **confirm_list),
                  OM_return_code *om_error);
```

Description

Registering for events means an application or agent registers with EMS to receive one or more events as they are generated. Calling OVeRegister means that the caller will receive all events generated at the specified list of nodes, limited only by the filter included with the call. The filter can be previously built with the OVeFilterAttr convenience routine.

OVeRegister accepts a list of network nodes specified by the IP address or the host name. Use of pattern-matching characters is not supported. Because the events will always be received at the local node, OVeRegister generates an mp_create_req to create a local event sieve with the caller as the destination address. If the events are to be captured on remote nodes, OVeRegister additionally generates mp_create_req commands for the remote nodes, creating sieves at each remote node. Only CMIP events are forwarded from remote nodes, even if the filter includes SNMP traps. All the event sieves use the filter definition, if any, included with the call.

Event sieve creation is a confirmed CMIS process, that is, EMS responds with sieve identification information or an error for each sieve created. OVeRegister returns the invoke IDs of the requests so that they can be matched with the event sieve confirmations returned from EMS. The confirmation notices must be received using the mp_receive function.

Parameters

session
Specifies an OM_private_object (returned from mp_bind) that identifies the binding established with the Communications Infrastructure.

workspace
Specifies the XOM workspace used in building the create requests.

node_list
Specifies a pointer to a linked list of destination node addresses where the event sieve should be created. If this list is empty, the sieve is created only on the local node. The local node may be

included in the list, but is not necessary; OVeRegister automatically builds an event sieve at the local node.

filter_attribute
Specifies a pointer to a structure returned by OVeFilterAttr that describes the filter to be applied with the sieve. If this parameter is NULL, no filter is specified, and all events will be forwarded.

confirm_list
Specifies a pointer to a linked list of OVeConvConfirm structures. The structures contain information for each create request that OVeRegister made: the destination node address, the invoke ID returned from the mp_create_req, and the status returned from mp_create_req. The caller needs to receive, through mp_receive, a confirmation of each create request that was made in order to ensure that each event sieve was created. Each successful confirmation will also contain the instance of the event sieve that was created. This instance is used by OVeDeregister to delete the event sieve when deregistering for events.

Return Values

If successful, OVeRegister returns 0 (zero). If unsuccessful, it returns one of the negative integers listed in the ove_xmp.h header file.

Files

When compiling a program that uses OVeRegister, you need to include the following files:

- xom.h
- xmp.h
- lnv.h (only if you are using any OVe calls)
- omp_dmi.h (only if you are using the dmi package)
- xmp_snmp.h
- xmp_cmis.h
- ove_xmp.h (only if you are using any OVe calls)
- ov_types.h

Libraries

When compiling a program that uses OVeRegister, you need to link to the following library:

/usr/OV/lib/libxmp.a

Related Information

- See “OVeDeregister(3)” on page 425.
- See “OVeFilterAttr(3)” on page 427.
- See “mp_create_req(3)” on page 49.
- See “mp_receive(3)” on page 76.

OVmib_get_objid_name(3)

Purpose

Converts a dotted-decimal MIB variable object identifier to its textual name

Syntax

```
#include <OV/OVsnmp.h>
const char *OVmib_get_objid_name(ObjectID *oid, u_int oid_length);
```

Description

OVmib_get_objid_name converts a dotted-decimal MIB object ID into its more meaningful textual name. If the textual name cannot be found, a character string version of the dotted-decimal object ID is returned. The following list shows example object IDs and the text strings that would be returned for them:

.1.3.6	.iso.org.dod
.0	.0

Return Values

OVmib_get_objid_name returns a pointer to the textual name of the specified MIB object ID.

Examples

An example program that shows how to use OVmib_get_objid_name is provided in the /usr/OV/prg_samples/nvsnmp_app/name_to_oid.c file.

Libraries

When compiling a program that uses OVmib_get_objid_name, you need to link to the following library:

- /usr/OV/lib/libmib.a

OVmib_read_objid(3)

Purpose

Converts a MIB variable name to its object identifier format

Syntax

```
#include <OV/OVsnmp.h>
int OVmib_read_objid(const char *name, ObjectID *oid, u_int *oid_length);
```

Description

OVmib_read_objid converts a textual MIB variable name into its equivalent dotted-decimal object ID format. While performing a MIB tree lookup, this routine uses the following names for the default prefix for the textual name:

.iso.org.dod.internet.mgmt.mib-2	Checked first
.iso.org.dod.internet.private.enterprises	Checked next if the name was not found in the first directory

OVmib_read_objid does not use a prefix if the name starts with a period (.).

This conversion provides sufficient mappings for the following sample MIB names:

system.sysContact	.1.3.6.1.2.1.1.4
ibm.ibmProd.netview6000	.1.3.6.1.4.1.2.6.3
.iso.org.dod	.1.3.6
iso.org.dod	Returns an error because there is no leading period and this variable does not exist relative to either of the preceding directories

Return Values

If successful, OVmib_read_objid returns 0 (zero). If unsuccessful, it returns -1, which means that the specified MIB variable name could not be found in the MIB tree.

Examples

An example program that shows how to use OVmib_read_objid is provided in the /usr/OV/prg_samples/nvsnmp_app/name_to_oid.c file.

Libraries

When compiling a program that uses OVmib_read_objid, you need to link to the following library:

- /usr/OV/lib/libmib.a

OVsnmpAddVarBind(3)

Purpose

Allocates space for and initializes an OVsnmpVarBind data structure for getting and setting variables

Related Functions

OVsnmpAddNullVarBind
OVsnmpAddTypedVarBind

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpVarBind *OVsnmpAddNullVarBind(OVsnmpPdu *pdu, ObjectID *oid, int oid_len);

OVsnmpVarBind *OVsnmpAddTypedVarBind(OVsnmpPdu *pdu, ObjectID *oid, int oid_len,
       u_char type, OVsnmpVal *val, int val_len)
```

Description

OVsnmpAddNullVarBind creates a new OVsnmpVarBind data structure and adds it to the OVsnmpPdu data structure pointed to by the pdu parameter. The oid (object identifier) and oid length fields in the new OVsnmpVarBind data structure are initialized. The ASN type is set to ASN_NULL, and the other fields are set to 0 (zero) or NULL as appropriate.

OVsnmpAddTypedVarBind also creates a new OVsnmpVarBind data structure and adds it to the OVsnmpPdu data structure pointed to by the pdu parameter. The oid (object identifier) and oid length fields in the new OVsnmpVarBind data structure are initialized and space is allocated for the value of the variable. The type, value, and value length for the variable are then assigned. This is useful when a management station is setting variables in an agent, and, in setting up these variables, has determined the type of variable being sent.

The memory allocated for the new OVsnmpVarBind data structure and the value of the variable are dynamic. They will be freed by a call to OVsnmpFreePdu, or, if the FREE_PDU bit is set in the session_flags variable of the sending session, by the OVsnmpSend or OVsnmpBlockingSend functions.

Parameters

pdu
Specifies a pointer to an OVsnmpPdu data structure returned by a call to OVsnmpCreatePdu. The new OVsnmpVarBind structure will be added to this PDU.

oid
Specifies a pointer to the object identifier value that will be assigned to this variable. This is generally the name of an array of type ObjectID.

oid_len
Specifies the number of elements in the oid. Note that this is not the number of bytes in the ObjectID variable. The maximum value for this parameter is MAX_SUBID_LEN.

type
Specifies the ASN.1 type that will be assigned to the variable. Valid types are provided in the <OVsnmpAsn1.h> header file.

val
Specifies a pointer to the value that will be assigned to the variable. Space will be allocated by OVsnmpAddTypedVarBind to hold the value of the variable.

val_len
Specifies the number of elements in the variable. For example, an ObjectID variable with the value .1.3.6.1.2.1 would have a length of 6 and an integer value would have a length of 1.

Return Values

If successful, OVsnmpAddVarBind returns a pointer to the new OVsnmpVarBind structure that was added to the OVsnmpPdu structure. If unsuccessful, it returns NULL.

Note: If successful, OVsnmpBlockingSend returns a pointer to an OVsnmpPdu structure that contains the response to the outbound PDU. If unsuccessful, it returns NULL.

Error Codes

OVsnmpAddVarBind returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_SYSERR_MALLOC]	The malloc system call failed. The global variable errno contains the malloc specific error.
[SNMP_ERR_BAD_LENGTH]	oid_len is <= zero or > MAX_SUBID_LEN.
[SNMP_ERR_BAD_TYPE]	The ASN.1 type of the variable was not valid.
[SNMP_ERR_BAD_VALUE]	The val and val_len parameters are incompatible. This can happen if the type of the value is INTEGER and val_len is not equal to one.

Libraries

When compiling a program that uses OVsnmpAddVarBind or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpCreatePdu(3)” on page 490.
- See “OVsnmpDoRetry(3)” on page 492.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpFreePdu(3)” on page 497.

OVsnmpBlockingSend(3)

Purpose

Sends an SNMP PDU and waits for the response

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpPdu *OVsnmpBlockingSend(OVsnmpSession *session, OVsnmpPdu *pdu)
```

Description

OVsnmpBlockingSend performs the required ASN.1 encoding on the specified PDU and sends the serialized PDU to the destination that was specified for the given session at the time the session was created. The calling process then blocks, with possible retransmissions of the pdu, until a response is received or a time-out occurs.

Time-outs can occur for a number of reasons, such as the destination host not providing SNMP services, congested networks, or the destination host is down. Time-outs do not indicate that an error in processing has occurred.

The time-out interval is specified using a combination of the retries and interval values that were set at session creation time. The PDU will be retransmitted according to the following rules:

Case 1: retries > 0 and interval > 0
The PDU will be retransmitted retries times with a wait of interval in tenths-of-seconds between the retransmissions. There is an exponential increase on the interval imposed by the library.

Case 2: retries = 0 and interval > 0
The PDU will not be retransmitted. The calling process will block until the response is received or for interval tenths-of-seconds.

Case 3: retries = 0 and interval = 0
The PDU will not be retransmitted. The calling process will block for DEFAULT_BLOCKING_TIMEOUT seconds or until a response is received.

If no response is received before the maximum number of retries is reached, a NULL pointer is returned to the calling process and the global variable OVsnmpErrno will be set to SNMP_ERR_NO_RESPONSE.

While the calling process is blocked it is possible for a response or trap to arrive for a session that is operating in non-blocking manner. When this happens, the inbound PDU will be delivered to the callback function registered in the non-blocking session.

If the OVsnmpBlockingSend call succeeds, the PDU specified in the pdu parameter is freed by a call to OVsnmpFreePdu. If the calling process needs to override this behavior, the FREE_PDU bit can be turned off in the session_flags variable in the session parameter. If this is done, the calling process must free the PDU with a call to OVsnmpFreePdu. If this is not done, the calling process will unnecessarily consume memory. The FREE_PDU bit is on by default.

Note: OVsnmpBlockingSend should not be used with sessions that were created using the OVsnmpXOpen call. OVsnmpXSend should be used instead.

Parameters

session
Specifies a pointer to a valid OVsnmpSession structure returned by a call to OVsnmpOpen. The OVsnmpSession structure is used to determine the destination, retry information and community name to be used in transmitting the SNMP PDU specified in the pdu parameter.

pdu
Specifies a pointer to a valid OVsnmpPdu structure returned by a call to OVsnmpCreatePdu. The pdu structure contains the PDU type and a pointer to the OVsnmpVarBind list.

Return Values

If successful, OVsnmpBlockingSend returns a pointer to an OVsnmpPdu structure that contains the response to the outbound PDU. If unsuccessful, it returns NULL.

Error Codes

OVsnmpBlocking returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_ERR_NO_RESPONSE] No response received before a time-out occurred.

[SNMP_ERR_BAD_SESSION] The session parameter does not point to an OVsnmpSession data structure that was created by OVsnmpOpen.

[SNMP_ERR_PDU_BUILD] An internal error occurred while ASN.1 was encoding the PDU. There might be a type that is not valid in one of the variables. This can happen if the OVsnmpVarBind data structure is modified after a call to OVsnmpAddTypedVarBind.

[SNMP_ERR_BAD_PDU_TYPE] The OVsnmpPdu data structure was not a get request, get next request, or a set request. This can happen if the OVsnmpPdu data structure is modified after a call to OVsnmpCreatePdu.

[SNMP_SYSERR_SENDTO] The sendto system call failed. The external variable errno contains the sendto specific error.

[SNMP_SYSERR_SELECT] The select system call failed. The external variable errno contains the select specific error.

[SNMP_SYSERR_MALLOC] The malloc system call failed. The external variable errno contains the malloc specific error.

Note: If OVsnmpBlockingSend is successful or an SNMP_ERR_NO_RESPONSE error occurs, and the FREE_PDU bit in the session_flags is turned on (default case), the request pdu parameter is freed by OVsnmpBlockingSend. The memory associated with the pdu parameter should not be referenced again. However, if OVsnmpBlockingSend returns another error, it does not free the memory for the pdu parameter. If the FREE_PDU bit in the session_flags has been explicitly turned off by the calling process, the memory associated with the pdu parameter is never freed by OVsnmpBlockingSend. The calling process must free the pdu with a call to OVsnmpFreePdu. If this is not done, the calling process will consume unneccessary amounts of memory.

Libraries

When compiling a program that uses OVsnmpBlockingSend, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpClose(3)” on page 440.
- See “OVsnmpCreatePdu(3)” on page 490.
- See “OVsnmpAddVarBind(3)” on page 435.
- See “OVsnmpSend(3)” on page 514.

OVsnmpClose(3)

Purpose

Ends an SNMP session and frees resources allocated by the session

Related Functions

OVsnmpXClose

Syntax

```
#include <OV/OVsnmp.h>

int OVsnmpClose(OVsnmpSession *session)
int OVsnmpXClose(OVsnmpSession *session)
```

Description

The OVsnmpClose function frees all memory used by the specified session, including pending requests, and closes the socket descriptor associated with this session. The specified session should not be referenced again.

The two functions OVsnmpClose and OVsnmpXClose perform the same basic functions; they both free all resources owned by session. However, OVsnmpClose must be used when session was created by a call to OVsnmpOpen. OVsnmpXClose must be used when session was created by OVsnmpXOpen. The OVsnmpXClose function is intended to be used with X11.

Parameters

session
Specifies a pointer to an OVsnmpSession structure that was allocated by a call to OVsnmpOpen or OVsnmpXOpen,

Return Values

If successful, OVsnmpClose and OVsmnpXClose return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVsnmpOpenClose and OVsnmpXClose return the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_ERR_BAD_SESSION]
Session was not created by OVsnmpOpen or OVsnmpXOpen.

Libraries

When compiling a program that uses OVsnmpClose or OVsnmpXClose, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpSend(3)” on page 514.

OVsnmpConfAllocEntry(3)

Purpose

Allocates dynamic storage for an OVsnmpConfEntry structure.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfAllocEntry( void );
```

Description

Allocates dynamic storage for an OVsnmpConfEntry structure.

Parameters

None.

Return Values

When successful, this function returns a non-null pointer to an OVsnmpConfEntry structure. If a failure occurs, a null pointer is returned, and OVsnmpErrno is set.

The memory pointed to by the return pointer is dynamically allocated and should be freed by the caller.

Error Codes

[SNMP_SYSERR_MALLOC] Storage cannot be allocated.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfFreeEntry(3)” on page 457

OVsnmpConfAllocWcList(3)

Purpose

Allocates dynamic storage for an OVsnmpConfWcList structure.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfWcList *  OVsnmpConfAllocWcList ( void );
```

Description

This function allocates dynamic storage for an OVsnmpConfWcList structure. The memory which is returned is dynamically allocated and should be freed by the caller.

Parameters

None.

Return Values

When successful, this function returns a non-null pointer to an OVsnmpWcList structure. If a failure occurs, a null pointer is returned, and OVsnmpErrno is set.

Error Codes

[SNMP_SYSERR_MALLOC] Storage cannot be allocated.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfFreeWcList(3)” on page 459

OVsnmpConfClose(3)

Purpose

Closes an SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfClose ( void );
```

Description

This function closes an SNMP Configuration Database. It frees all internal storage associated with an open database.

Parameters

None.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfOpen(3)” on page 460

OVsnmpConfCopyEntry(3)

Purpose

Allocates a new OVsnmpConfEntry and copies the contents of the old OVsnmpConfEntry to the new one.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfCopyEntry ( OVsnmpConfEntry *ce );
```

Description

This function allocates dynamic storage for a new OVsnmpConfEntry structure. The contents of the OVsnmpConfEntry structure pointed to by the parameter *ce* are copied to the new OVsnmpConfEntry. All character strings are duplicated in the new structure.

The memory pointed to by the return pointer is dynamically allocated and should be freed by the caller. The OVsnmpConfFreeEntry(3) function should be used.

Parameters

ce A pointer to the OVsnmpConfEntry structure whose contents are to be copied.

Return Values

If successful, a non-null pointer to an OVsnmpConfEntry structure is returned. Otherwise, a null pointer is returned.

Error Codes

[SNMP_SYSERR_MALLOC] Storage cannot be allocated.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfFreeEntry(3)” on page 457
- ovsnmp.conf(4)

OVsnmpConfCreateEntry(3)

Purpose

Creates a configuration record in the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfCreateEntry ( OVsnmpConfEntry *ce );
```

Description

This function creates a configuration record in the SNMP Configuration Database. The configuration data is obtained from the input OVsnmpConfEntry structure. It uses the *name* field of this structure to determine if a conflicting record already exists. If there is a conflict, no record is created, and an error is returned. It also checks that the fields of the OVsnmpConfEntry structure contain consistent and valid data.

Note that if the database is modified all cached configuration information is deleted.

Parameters

ce A pointer to an OVsnmpConfEntry structure, whose contents are to be stored in the SNMP Configuration Database.

For more information on the OVsnmpConfEntry data structure, see “OVsnmpIntro(5)” on page 501 or *NetView for AIX Programmer's Guide*.

Return Values

This function returns 0 if successful, and -1 if failure.

Error Codes

[SNMP_SYSERR_MALLOC] Internal memory allocation failed.

[SNMP_ERR_DB_NOT_OPEN] The database has not been opened.

[SNMP_ERR_DB_WRITE_ERROR] The database cannot be written.

[SNMP_ERR_DB_OVERWRITE_ERROR]
A conflicting record already exists in the SNMP Configuration Database.

[SNMP_ERR_DB_CORRUPTED_CACHE]
Cached information cannot be deleted after the SNMP Configuration Database has been updated.

[SNMP_ERR_INVALIDHOST] Either the "name" field or the "proxy" field in the OVsnmpConfEntry structure cannot be resolved to a valid IP address.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
An invalid remote port is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
An invalid poll interval is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_TIMEOUT]
An invalid timeout period is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_RETRY] An invalid number of retries is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_COLONS_IN_STRING]
Any of the character strings in the OVsnmpConfEntry structure contain a ":". This restriction is required for backward compatibility with the Version 2 ovsnmp.conf file.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The community name or the setCommunity name in the OVsnmpConfEntry structure exceed MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_NAME] The name field of the OVsnmpConfEntry structure contains characters that may be construed as wildcard characters, or is null.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreDefault(3)” on page 486
- “OVsnmpConfStoreEntry(3)” on page 488
- ovsnmp.conf(4)

OVsnmpConfDbName(3)

Purpose

Determines the name of the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

const char *  OVsnmpConfDbName( void )
```

Description

This function returns a pointer to the name of the SNMP Configuration Database. If the database is currently open, the name of the open database is returned. Otherwise, the environment variable, OVSNMP_CONF_FILE is read. If this variable is set, the return value points to a string which is the concatenation of the pathname specified in OVSNMP_CONF_FILE together with the suffix "_db". If OVSNMP_CONF_FILE is not set, the default database name, /usr/OV/conf/ovsnmp.conf_db, is returned.

Parameters

None.

Return Values

This routine returns a pointer to a character string containing the name of the SNMP Configuration Database. This pointer points to static storage, and should NOT be freed.

Error Codes

None.

Dependencies

The environment variable OVSNMP_CONF_FILE is maintained for backward compatibility with Version 2. The pathname of the database is obtained from this variable (when it is set) by appending the suffix "_db" to the variable pathname.

OVsnmpConfDeleteCache(3)

Purpose

Removes all cached SNMP configuration data from an open database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfDeleteCache( void );
```

Description

This function removes all cached SNMP configuration data from the persistent cache. The cached data will normally accumulate and persist across processes. The level and type of caching can be set by the OVsnmpConfStoreCntl(3) function. See ovsnmp.conf(4) for more details.

Parameters

None.

Return Values

0 if successful, or the database is not open; -1 on failure.

Error Codes

[SNMP_ERR_DB_CORRUPTED_CACHE]
The deletion cannot be done and the database is open.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreCntl(3)” on page 484
- ovsnmp.conf(4)

OVsnmpConfDeleteEntry(3)

Purpose

Deletes a record from the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfDeleteEntry (  char *key );
```

Description

This function deletes the configuration record for the target indicated by *key* from the SNMP Configuration Database. First a deletion is attempted using the literal *key* argument to the function. Next, the *key* is resolved to a fully qualified IP domain name, and the deletion is attempted with this new "key".

If the deletion was successful, the persistent cache of configuration data is removed. Also note that if no record corresponds to the key, the delete operation returns successfully, and the cache is removed.

Parameters

key
: A pointer to a character string containing the name of the target whose configuration information is to be deleted.

Return Values

0 if successful, -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN] The SNMP Configuration Database has not been opened.

[SNMP_ERR_DB_WRITE_ERROR] The caller is unable to write to the SNMP Configuration Database.

[SNMP_ERR_DB_CORRUPTED_CACHE] The persistent cache of SNMP Configuration Data cannot be removed.

Dependencies

OVsnmpConfOpen(3) must be called prior to this call.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfOpen(3)” on page 460
- ovsnmp.conf(4)

OVsnmpConfExportFile(3)

Purpose

Dumps the contents of the SNMP Configuration Database to a file.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfExportFile ( const char *filename,
                            exportFlags_t flags );
```

Description

This function dumps the contents of the SNMP Configuration Database to a file. Unless the SNMP_CONF_EXPORT_VERBOSE flag is specified, the file is in a form compatible with the Version 2 ovsnmp.conf(4) file and can be used by applications bound with the Version 2 SNMP library.

Parameters

filename
A pointer to a character string indicating the file into which the contents of the SNMP Configuration Database are to be dumped.

flags
A set of options which may be OR-ed together in order to limit the amount and kind of configuration data that is to be dumped.

The flags options are:

SNMP_CONF_EXPORT_ALL	Dump all configuration records.
SNMP_CONF_EXPORT_WCLIST	Dump only wildcarded configurations records.
SNMP_CONF_EXPORT_32ONLY	Dump only configurations records that are compatible with Version 2 configuration parameters.
SNMP_CONF_EXPORT_VERBOSE	Dump configuration records in a more readable format. This format CANNOT be used for subsequent input to the OVsnmpConfImportFile(3) function nor can it be used by applications bound with the Version 2 SNMP library.
SNMP_CONF_EXPORT_SHADOW	Dump configuration records according to the compatibility mode specified in the database control record. See “OVsnmpConfReadCntl(3)” on page 470 for details about this control record.

Return Values

0 if successul; -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.
[SNMP_ERR_DB_NO_WRITE_PERM]	The file to which the SNMP Configuration Database will be dumped is unwriteable.

Warning

If the resulting file is to be used by applications which are bound with the Version 2 SNMP library, or the file is to be used as subsequent input to the OVsnmpConfImportFile(3) function, do NOT use the SNMP_CONF_EXPORT_VERBOSE flag.

Dependencies

OVsnmpConfOpen(3) must be called before using this function.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadCntl(3)” on page 470
- “OVsnmpConfImportFile(3)” on page 463
- ovsnmp.conf(4)

OVsnmpConfFileName(3)

Purpose

Determines the pathname of the Version 2 backward-compatibility SNMP configuration file associated with the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

const char *  OVsnmpConfFileName ( void );
```

Description

This function is used to determine the pathname of the Version 2 backward-compatibility SNMP configuration file. This file is associated with the SNMP Configuration Database. When database shadowing is enabled, the contents of the database are stored in this file in a format that is readable by applications which are bound with the Version 2 SNMP Library.

The default pathname is /usr/OV/conf/ovsnmp.conf. This can be overridden by the environment variable OVSNMP_CONF_FILE.

A discussion of database shadowing can be found in ovsnmp.conf(4).

Parameters

none

Return Values

This routine returns a pointer to a static character string which contains the pathname of the Version 2 backward-compatibility SNMP configuration file. This storage should NOT be freed by the caller.

Error Codes

None.

Dependencies

The environment variable OVSNMP_CONF_FILE is maintained for backward compatibility with Version 2. The pathname of the database is obtained from this variable (when it is set) by appending the suffix "_db" to the variable pathname.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadCntl(3)” on page 470
- “OVsnmpConfStoreCntl(3)” on page 484
- “OVsnmpConfDbName(3)” on page 448
- ovsnmp.conf(4)

OVsnmpConfFreeDest(3)

Purpose

Frees an OVsnmpConfDest structure and its contents.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfFreeDest ( OVsnmpConfDest  *dd );
```

Description

This function frees an OVsnmpConfDest structure (and its contents) that had been allocated by the SNMP library. Only the contents of non-null structure pointers is freed.

Parameters

dd A pointer to the OVsnmpConfDest structure that is to be freed. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfResolveDest(3)” on page 482

OVsnmpConfFreeEntry(3)

Purpose

Frees an OVsnmpConfEntry structure and its contents.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfFreeEntry ( OVsnmpConfEntry  *ce );
```

Description

This function frees an OVsnmpConfEntry structure (and its contents) which have been allocated by one of the following routines:

- OVsnmpConfReadEntry(3),
- OVsnmpConfReadNextEntry(3),
- OVsnmpConfCopyEntry(3),
- OVsnmpConfParseEntry(3),
- OVsnmpConfAllocEntry(3),
- OVsnmpConfReadDefault(3)

Parameters

ce A pointer to the OVsnmpConfEnty structure that is to be freed. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadEntry(3)” on page 474
- “OVsnmpConfReadNextEntry(3)” on page 478
- “OVsnmpConfCopyEntry(3)” on page 445
- “OVsnmpConfParseEntry(3)” on page 465
- “OVsnmpConfAllocEntry(3)” on page 442
- “OVsnmpConfReadDefault(3)” on page 472
- ovsnmp.conf(4)

OVsnmpConfFreeWcList(3)

Purpose

Frees an OVsnmpConfWcList structure and its contents.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfFreeWcList (  OVsnmpConfWcList  *wc );
```

Description

This function frees an OVsnmpConfWcList structure (and its contents) which was allocated by OVsnmpConfReadWcList(3) or OVsnmpConfAllocWcList(3).

This routine frees only the contents of non-null structure pointers.

Parameters

wc A pointer to the OVsnmpConfWcList structure that is to be freed. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadWcList(3)” on page 480
- “OVsnmpConfAllocWcList(3)” on page 443
- ovsnmp.conf(4)

OVsnmpConfOpen(3)

Purpose

Opens a SNMP Configuration database for subsequent use.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfOpen ( openFlags_t flags );
```

Description

This function opens a SNMP Configuration database. All subsequent calls to functions which access the database require that the database be opened.

The default database which will be opened is /usr/OV/conf/ovsnmp.conf_db. For backward compatibility with Version 2, this location can be changed using the environment variable OVSNMP_CONF_FILE. The pathname of the database is derived from this variable by appending the suffix "_db" to the variable, that is, ${OVSNMP_CONF_FILE}_db will be the name of the database that is opened.

OVSNMP_CONF_FILE is also the name of the Version 2 compatible shadow file when shadowing is enabled.

Parameters

flags
A set of options which may be OR-ed together as appropriate to control the behavior of the open function.

The following flag values are available:

SNMP_CONF_OPEN_RDONLY	Open the database for reading only.
SNMP_CONF_OPEN_RDWR	Open the database for reading and writing.
SNMP_CONF_OPEN_TRUNC	Open the database and delete all stored configuration data.
SNMP_CONF_OPEN_CREATE	Create the database if necessary.
SNMP_CONF_OPEN_NO_3_2	Do not maintain the Version 2 compatible file if the database is modified. This flag overrides whatever is specified in the database control record concerning Version 2 compatibility. See “OVsnmpConfReadCntl(3)” on page 470 for details.

Return Values

0 if successul; -1 if failure.

Error Codes

[SNMP_SYSERR_MALLOC]	internal memory allocation failure.
[SNMP_ERR_DB_NO_WRITE_PERM]	The database access permissions do not allow opening the database for writing.

[SNMP_ERR_DB_CANNOT_CREATE] The directory permissions for the directory containing the database do not permit the creation of the database.

[SNMP_ERR_DB_DOES_NOT_EXIST]
The database cannot be opened because it does not exist.

[SNMP_SYSERR] A system error occurred.

[SNMP_ERR_DB_READ_ERROR] The database cannot be read, either due to lack of permission or corruption.

When an attempt to create a temporary database fails, the following additional error codes may be returned:

[SNMP_ERR_DB_NO_READ_PERM] No permission to read the Version 2 compatible configuration file from which the database will be created.

[SNMP_ERR_DB_INVALID_TIMEOUT]
The Version 2 compatible file contains an entry with an invalid timeout value.

[SNMP_ERR_DB_INVALID_RETRY] The Version 2 compatible file contains an entry with an invalid retry value.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
The Version 2 compatible file contains an entry with an invalid poll interval value.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
The Version 2 compatible file contains an entry with an invalid remote port value.

[SNMP_ERR_DB_INVALID_NAME] The Version 2 compatible file contains an entry with an invalid target name.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The Version 2 compatible file contains an entry with a community string with greater than MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
The Version 2 compatible file contains an entry with an invalid wildcard specification.

[SNMP_ERR_DB_OVERWRITE_ERROR]
The Version 2 compatible file contains conflicting entries.

[SNMP_ERR_DB_WRITE_ERROR] The database cannot be written.

[SNMP_ERR_INVALIDHOST] The Version 2 compatible file contains an entry with an invalid destination, that is, the destination cannot be resolved to an IP address.

Warning

Caution should be used when storing files in the location pointed to by OVSNMP_CONF_FILE, or in the default location of the Version 2 compatible shadow file, /usr/OV/conf/ovsnmp.conf. When shadowing is enabled, these files can be modified.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadCntl(3)” on page 470
- “OVsnmpConfClose(3)” on page 444
- “OVsnmpConfImportFile(3)” on page 463
- ovsnmp.conf(4)

OVsnmpConfImportFile(3)

Purpose

Replaces the contents of the SNMP Configuration Database with configuration information obtained from a Version 2 compatible configuration file.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfImportFile( const char *filename,
                           importFlags_t flag );
```

Description

This function replaces the contents of the SNMP Configuration Database with configuration information obtained from a Version 2 compatibile configuration file. The form of this file is described in ovsnmp.conf(4) The wildcard features and semantics provided by this file in Version 2 are retained.

Parameters

filename
A pointer to a character string which contains the pathname of the file from which the database information will be imported.

flag
One of a set of options which may be used to influence the behavior or the import function.

The flags options are:

SNMP_CONF_IMPORT_ALL	Import all configuration entries from the file into the database.
SNMP_CONF_IMPORT_WCLIST	Import only wildcard configuration entries from the file into the database.
SNMP_CONF_IMPORT_CHECK	Only check that the imported file contains valid entries.

Return Values

0 if success; -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database has not previously been opened.
[SNMP_ERR_DB_NO_READ_PERM]	The caller does not have read permission for the import file.
[SNMP_SYSERR_MALLOC]	Internal memory allocation failed.
[SNMP_ERR_DB_INVALID_TIMEOUT]	A configuration entry in the import file contains an invalid timeout value.
[SNMP_ERR_DB_INVALID_RETRY]	A configuration entry in the import file contains an invalid retry value.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
A configuration entry in the import file contains an invalid poll interval.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
A configuration entry in the import file contains an invalid remote port value.

[SNMP_ERR_DB_INVALID_NAME] A configuration entry in the import file contains an invalid target name.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
A configuration entry in the import file contains a community string with greater than MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
A configuration entry in the import file contains an invalid wildcard specification.

[SNMP_ERR_DB_OVERWRITE_ERROR]
The import file contains conflicting configuration entries.

[SNMP_ERR_DB_WRITE_ERROR] The SNMP Configuration Database cannot be written.

[SNMP_ERR_INVALIDHOST] A configuration entry in the import file contains an invalid destination.

[SNMP_ERR_DB_CORRUPTED_CACHE]
The cached SNMP Configuration Database data cannot be removed.

[SNMP_ERR_DB_READ_ERROR] The cached SNMP Configuration Database cannot be read.

Warning

This function causes the cache of SNMP configuration data to be deleted.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfExportFile(3)” on page 452
- “OVsnmpConfParseEntry(3)” on page 465
- ovsnmp.conf(4)

OVsnmpConfParseEntry(3)

Purpose

Parses a line in Version 2 ovsnmp.conf file form and produces an OVsnmpConfEntry structure.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfParseEntry ( char *line );
```

Description

This function parses a line in Version 2 ovsnmp.conf file form and produces an OVsnmpConfEntry structure. See ovsnmp.conf(4) for details on the format of these lines. This format has been extended to include two additional colon-separated fields, the remote port field and the setCommunity name, respectively.

The returned pointer points to dynamically allocated storage. This storage should be eventually freed using OVsnmpConfFreeEntry(3).

Parameters

line
: A pointer to a character string which contains a Version 2 ovsnmp.conf file, colon-separated configuration string. This string may contain two appended fields, the remote port field and the setCommunity name, respectively.

Return Values

This routine returns a pointer to an OVsnmpConfEntry structure with all fields initialized according to the values supplied in the input line. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Error Codes

[SNMP_SYSERR_MALLOC] internal memory allocation failure.

[SNMP_ERR_DB_INVALID_TIMEOUT]
: *line* contains an entry with an invalid timeout value.

[SNMP_ERR_DB_INVALID_RETRY] *line* contains an entry with an invalid retry value.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
: *line* contains an entry with an invalid poll interval value.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
: *line* contains an entry with an invalid remote port value.

[SNMP_ERR_DB_INVALID_NAME] *line* contains an entry with an invalid target name.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
: *line* contains an entry with a community string with greater than MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
: *line* contains an entry with an invalid wildcard specification.

[SNMP_ERR_INVALIDHOST]
: *line* contains an entry with an invalid destination, that is, the the destination cannot be resolved to an IP address.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfImportFile(3)” on page 463
- ovsnmp.conf(4)

OVsnmpConfPrintCntl(3)

Purpose

Prints the database control information to stdout.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfPrintCntl ( OVsnmpConfCntl *cc );
```

Description

This function prints the database control information to stdout. This information can be obtained using the function OVsnmpConfReadCntl(3).

Parameters

cc A pointer to an OVsnmpConfCntl structure that contains the database control information. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadCntl(3)” on page 470
- ovsnmp.conf(4)

OVsnmpConfPrintDest(3)

Purpose

Prints the resolved SNMP configuration parameters for the target destination to stdout.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfPrintDest ( OVsnmpConfDest *dd );
```

Description

This function prints the resolved SNMP configuration parameters for the target destination to stdout. This information can be obtained using the function OVsnmpConfResolveDest(3).

Parameters

dd A pointer to an OVsnmpConfDest structure that contains the resolved SNMP configuration parameters for the target destination. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfResolveDest(3)” on page 482
- ovsnmp.conf(4)

OVsnmpConfPrintEntry(3)

Purpose

Prints the SNMP configuration parameters for a target, wildcard, or global default to stdout.

Syntax

```
#include  <OV/OVsnmpConf.h>

void  OVsnmpConfPrintEntry ( OVsnmpConfEntry *ce );
```

Description

This function prints the SNMP configuration parameters for a particular target, a wildcard, or the global default to stdout. This information can be obtained using the functions OVsnmpConfReadEntry(3), OVsnmpConfReadWcList(3), or OVsnmpConfReadDefault(3) as appropriate.

Parameters

ce A pointer to an OVsnmpConfEntry structure that contains the target wildcard, or global default SNMP configuration parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

Return Values

None.

Error Codes

None.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfReadEntry(3)” on page 474
- “OVsnmpConfReadWcList(3)” on page 480
- “OVsnmpConfReadDefault(3)” on page 472
- ovsnmp.conf(4)

OVsnmpConfReadCntl(3)

Purpose

Reads the control parameters of the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfCntl *  OVsnmpConfReadCntl ( void );
```

Description

This function returns the control parameters of the SNMP Configuration Database. These configurable control parameters are described in ovsnmp.conf(4) and xnmsnmpconf(1). The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller.

Parameters

None.

Return Values

This routine returns pointer to a dynamically allocated OVsnmpConfCntl structure which contains the SNMP Configuration Database control parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

A null pointer is returned if a failure occurs.

Error Codes

[SNMP_SYSERR_MALLOC]	Storage cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreCntl(3)” on page 484
- “OVsnmpConfPrintCntl(3)” on page 467
- ovsnmp.conf(4)

OVsnmpConfReadDefault(3)

Purpose

Reads the global default parameters in the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfReadDefault ( void );
```

Description

This function returns the global default parameters in the SNMP Configuration Database. If no global default has been explicitly stored in the database, the hard-coded system default parameters are returned. These parameters are described in ovsnmp.conf(4).

Parameters

None.

Return Values

This routine returns a pointer to a dynamically allocated OVsnmpConfEntry structure which contains the SNMP Configuration Database default parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

A null pointer is returned if a failure occurs.

Error Codes

[SNMP_SYSERR_MALLOC]	Memory cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read. The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeEntry(3) function.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreDefault(3)” on page 486
- “OVsnmpConfFreeEntry(3)” on page 457
- ovsnmp.conf(4)

OVsnmpConfReadEntry(3)

Purpose

Reads the parameters for the target node from the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfReadEntry ( char * key );
```

Description

This function returns SNMP Configuration parameters for the target node specified by the *key* argument. It does NOT resolve defaulted parameters. These parameters are described in ovsnmp.conf(4).

The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeEntry(3) function.

Parameters

key

A pointer to a character string that contains the name by which the target node parameters are looked up. If the configuration parameters were stored with an IP address string target name, they must be looked up using this string. If they were stored using a domain name, they may be accessed by the fully qualified domain name, or a suitable alias. Proxied nodes must be accessed by the target name that was used when the parameters were stored in the database.

Return Values

This routine returns a pointer to a dynamically allocated OVsnmpConfEntry structure which contains the SNMP Configuration Database parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure. A null pointer is returned if there is no entry that corresponds to *key*, or if a failure occurs.

Error Codes

[SNMP_ERR_NOERROR]	There is no entry that corresponds to the key.
[SNMP_SYSERR_MALLOC]	Memory cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreEntry(3)” on page 488
- “OVsnmpConfFreeEntry(3)” on page 457
- ovsnmp.conf(4)

OVsnmpConfReadNextDest(3)

Purpose

Reads the next configuration entry from the SNMP Configuration Database cache.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfDest *  OVsnmpConfReadNextDest ( int flag );
```

Description

This function returns the next SNMP configuration entry from the SNMP Configuration Database cache. The first time the function is called, it will return the first entry in the database cache, regardless of the value of the *flag* argument.

The parameters in the returned entry are fully resolved, that is, the wildcard entries, global default, and hard-coded system defaults have been applied to defaulted parameters. These parameters are described in ovsnmp.conf(4).

The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeDest(3) function.

Parameters

flag

This parameter can be either of two values:

- SNMP_CONF_READ_FIRST, which resets the "next pointer" to the first entry and reads this entry.
- SNMP_CONF_READ_NEXT, which reads the next extry from the database.

Return Values

This routine returns a pointer to a dynamically allocated OVsnmpConfDest structure which contains the SNMP Configuration Database parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure.

A null pointer is returned if a failure occurs, or if there are no more entries to be read.

Error Codes

[SNMP_ERR_NOERROR]	There are no more entries to be read.
[SNMP_SYSERR_MALLOC]	Memory cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfResolveDest(3)” on page 482
- “OVsnmpConfFreeDest(3)” on page 456
- ovsnmp.conf(4)

OVsnmpConfReadNextEntry(3)

Purpose

Reads the next configuration entry from the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfEntry *  OVsnmpConfReadNextEntry ( int flag );
```

Description

This function returns the next SNMP configuration entry from the SNMP Configuration Database. The first time the function is called, it will return the first entry in the database (regardless of the value of the *flag* argument.) It does NOT resolve defaulted parameters. These parameters are described in ovsnmp.conf(4).

The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeEntry(3) function.

Parameters

flag

This parameter can be either of two values:

- SNMP_CONF_READ_FIRST, which resets the "next pointer" to the first entry and reads this entry.
- SNMP_CONF_READ_NEXT, which read the next extry from the database.

Return Values

This routine returns a pointer to a dynamically allocated OVsnmpConfEntry structure which contains the SNMP Configuration Database parameters. See “OVsnmpIntro(5)” on page 501 for the definition of this structure. A null pointer is returned if a failure occurs, or if there are no more entries to be read.

Error Codes

[SNMP_ERR_NOERROR]	There are no more entries to be read.
[SNMP_SYSERR_MALLOC]	Memory cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreEntry(3)” on page 488
- “OVsnmpConfFreeEntry(3)” on page 457
- ovsnmp.conf(4)

OVsnmpConfReadWcList(3)

Purpose

Reads the wildcard entries from the SNMP Configuration Database as a singly linked list.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfWcList * OVsnmpConfReadWcList ( void );
```

Description

This function returns the SNMP Configuration Database wildcards as a singly linked list. This sequence is returned in the order in which the wildcards would be applied when resolving the parameters for a particular target. This list and these parameters are described in ovsnmp.conf(4).

The list to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeWcList(3) function.

Parameters

None.

Return Values

This routine returns pointer to a dynamically allocated singly linked list of OVsnmpConfWcList structures which constitute the logical wildcard list contained in the SNMP Configuration Database. See “OVsnmpIntro(5)” on page 501 for the definition of these structures.

A null pointer is returned if there are no wildcards or if a failure occurs.

Error Codes

[SNMP_ERROR_NO_ERROR]	There are no wildcards in the database.
[SNMP_SYSERR_MALLOC]	Memory cannot be allocated.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_ERR_DB_READ_ERROR]	The SNMP Configuration Database cannot be read.

Dependencies

The database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreEntry(3)” on page 488
- “OVsnmpConfFreeWcList(3)” on page 459
- ovsnmp.conf(4)

OVsnmpConfResolveDest(3)

Purpose

Returns the resolved SNMP configuration parameters for a target node.

Syntax

```
#include  <OV/OVsnmpConf.h>

OVsnmpConfDest *  OVsnmpConfResolveDest ( char *key,
"                       "resolveFlags_t flags );
```

Description

This function returns the resolved SNMP configuration parameters for a target node. It consults the SNMP Configuration Database for specifically configured parameters for the target. It then applies the database wildcards to any defaulted fields. If there are still defaulted fields, the global default is applied, followed by the hard-coded system defaults.

The return pointer refers to dynamically allocated memory which must be freed by the caller using OVsnmpConfFreeDest(3).

Parameters

key
A pointer to a character string containing the identifier for the target node whose parameters are requested. For non-proxied IP targets this may be the fully qualified domain name, an alias known to the system name server, or a dotted IP address string. For proxied nodes, this string must be the target name that is configured for that node.

flags
This parameter can take on either of two values:

- SNMP_CONF_FORCE_RESOLVE, which causes the function to use whatever information is available to resolve the SNMP configuration parameters. If no other resolution can be made, it will return the hard-coded system defaults. It will return an error only if there is a problem allocating dynamic storage.
- SNMP_CONF_RESOLVE, which will cause an error to be returned in case of SNMP Configuration Database access or reading errors, or any other errors encountered.

Return Values

This routine returns a pointer to a dynamically allocated OVsnmpConfDest structure when successful. A null pointer is returned in case of failure.

Error Codes

[SNMP_ERR_INVALIDHOST]	The target indicated by *key* has no valid destination (that is, it has no valid IP address, or if it is proxied, the proxy has no valid IP address.), or is null.
[SNMP_ERR_DB_READ_ERROR]	The database cannot be read.

[SNMP_ERR_GETHOSTBYNAME]	A name server lookup failed.
[SNMP_ERR_DB_NOT_OPEN]	The SNMP Configuration Database is not open.
[SNMP_SYSERR_MALLOC]	Dynamic storage cannot be allocated.

Dependencies

The SNMP Configuration Database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfOpen(3)” on page 460
- “OVsnmpConfFreeDest(3)” on page 456
- ovsnmp.conf(4)

OVsnmpConfStoreCntl(3)

Purpose

Stores the control parameters for the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfStoreCntl (  OVsnmpConfCntl *cc );
```

Description

This function stores the control parameters for the SNMP Configuration Database. These parameters are stored in non-volatile storage and determine the behavior of the database for all processes that access it.

For more information on the OVsnmpConfEntry data structure, see “OVsnmpIntro(5)” on page 501 or *NetView for AIX Programmer's Guide*.

Return Values

0 if successful; -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN] The database has not been opened.

[SNMP_SYSERR_DB_MALLOC] An inter memory allocation fails.

[SNMP_ERR_DB_WRITE_ERROR] The database cannot be written.

[SNMP_ERR_DB_CORRUPTED_CACHE]
Cached information cannot be deleted after the SNMP Configuration Database has been updated.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
An invalid remote port is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
An invalid poll interval is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_TIMEOUT]
An invalid timeout period is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_RETRY] An invalid number of retries is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_COLONS_IN_STRING]
One of the character strings in the OVsnmpConfEntry structure contains a colon (":"). This restriction is required for backward compatibility with the Version 2 ovsnmp.conf file.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The community name or the setCommunity name in the OVsnmpConfEntry structure exceed MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
The name field of the OVsnmpConfEntry structure is not "*.*.*.*" or is null.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreEntry(3)” on page 488
- ovsnmp.conf(4)

OVsnmpConfStoreDefault(3)

Purpose

Stores the global default SNMP configuration parameters in the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfStoreDefault ( OVsnmpConfEntry *ce );
```

Description

This function stores the global default SNMP configuration parameters in the SNMP Configuration Database. The global default parameters are applied during a OVsnmpConfResolveDest(3) operation after wildcards have been applied.

Parameters

e A pointer to an OVsnmpConfEntry structure which contains the global default parameters.

The OVsnmpConfEntry is described in “OVsnmpIntro(5)” on page 501.

Return Values

0 if successful; -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN] The database has not been opened.

[SNMP_SYSERR_DB_MALLOC] An inter memory allocation fails.

[SNMP_ERR_DB_WRITE_ERROR] The database cannot be written.

[SNMP_ERR_DB_CORRUPTED_CACHE]
Cached information cannot be deleted after the SNMP Configuration Database has been updated.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
An invalid remote port is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
An invalid poll interval is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_TIMEOUT]
An invalid timeout period is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_RETRY] An invalid number of retries is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_COLONS_IN_STRING]
One of the character strings in the OVsnmpConfEntry structure contain a colon (":"). This restriction is required for backward compatibility with the Version 2 ovsnmp.conf file.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The community name or the setCommunity name in the OVsnmpConfEntry structure exceed MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
The name field of the OVsnmpConfEntry structure is not "*.*.*.*" or is null.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfStoreEntry(3)” on page 488
- ovsnmp.conf

OVsnmpConfStoreEntry(3)

Purpose

Stores the SNMP configuration parameters for a target in the SNMP Configuration Database.

Syntax

```
#include  <OV/OVsnmpConf.h>

int  OVsnmpConfStoreEntry ( OVsnmpConfEntry *ce );
```

Description

This function stores the SNMP configuration parameters for a target in the SNMP Configuration Database. These parameters are applied during an OVsnmpConfResolveDest(3) operation after wildcards have been applied.

For more information on the OVsnmpConfEntry data structure, see “OVsnmpIntro(5)” on page 501 or *NetView for AIX Programmer's Guide*.

Parameters

ce A pointer to an OVsnmpConfEntry structure which contains the parameters for the target.

Return Values

0 if successful; -1 if failure.

Error Codes

[SNMP_ERR_DB_NOT_OPEN] The database has not been opened.

[SNMP_SYSERR_MALLOC] An internal memory allocation fails.

[SNMP_ERR_DB_WRITE_ERROR] The database cannot be written.

[SNMP_ERR_DB_CORRUPTED_CACHE]
Cached information cannot be deleted after the SNMP Configuration Database has been updated.

[SNMP_ERR_INVALIDHOST] if, when proxying, the name does not resolve to an IP address, or if the proxy does not resolve to an IP address.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
An invalid remote port is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
An invalid poll interval is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_TIMEOUT]
An invalid timeout period is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_INVALID_RETRY] An invalid number of retries is specified in the OVsnmpConfEntry structure.

[SNMP_ERR_DB_COLONS_IN_STRING]
One of the character strings in the OVsnmpConfEntry structure contain a colon (":"). This restriction is required for backward compatibility with the Version 2 ovsnmp.conf file.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The community name or the setCommunity name in the OVsnmpConfEntry structure exceed MAX_COMMUNITY_LEN (255) characters.

Dependencies

The SNMP Configuration Database must be opened with OVsnmpConfOpen(3) before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- “OVsnmpConfOpen(3)” on page 460
- ovsnmp.conf(4)

OVsnmpCreatePdu(3)

Purpose

Allocates a OVsnmpPdu data structure of the specified message type

Syntax

```
#include <OV/OVsnmp.h>
OVsnmpPdu *OVsnmpCreatePdu(int type)
```

Description

The OVsnmpCreatePdu routine creates a new OVsnmpPdu data structure and initializes it for sending PDUs of the specified type. The memory associated with the new OVsnmpPdu is dynamically allocated. You should free it with a call to OVsnmpFreePdu, unless it is freed by OVsnmpSend or OVsnmpBlockingSend with FREE_PDU set.

SNMP PDUs are represented using a combination of the OVsnmpPdu data structure and the OVsnmpVarBind data structure. You should first create an OVsnmpPdu structure of a specific type using the OVsnmpCreatePdu call and add variables to the OVsnmpPdu structure with calls to either the OVsnmpAddNullVarBind or the OVsnmpAddTypedVarBind routine.

Parameters

type
Specifies the SNMP message type to associate with the new OVsnmpPdu data structure. The value of type must be one of the following:

- GET_REQ_MSG
- GETNEXT_REQ_MSG
- SET_REQ_MSG
- TRAP_REQ_MSG

Return Values

If successful, OVsnmpCreatePdu returns a pointer to a dynamically allocated OVsnmpPdu data structure. If unsuccessful, it returns NULL.

Error Codes

OVsnmpCreatePdu returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_ERR_BAD_PDU_TYPE]
The type parameter was not a valid SNMP PDU type.

[SNMP_SYSERR_MALLOC] The malloc system call failed. The global variable errno contains the malloc specific error.

Libraries

When compiling a program that uses OVsnmpCreatePdu, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpFixPdu(3)” on page 495.
- See OVsnmpAddNullVarBind in “OVsnmpAddVarBind(3)” on page 435.
- See OVsnmpAddTypedVarBind in “OVsnmpAddVarBind(3)” on page 435.
- See “OVsnmpFreePdu(3)” on page 497.

OVsnmpDoRetry(3)

Purpose

Retransmits pending SNMP requests

Syntax

```
#include  <OV/OVsnmp.h>
#include  <sys/time.h>

void OVsnmpDoRetry( )
```

Description

OVsnmpDoRetry is intended to be used with OVsnmpGetRetryInfo and select when you are using the ovsnmp or the nvsnmp library in a non-blocking manner. The calling process uses OVsnmpDoRetry to retransmit pending SNMP requests. For coding examples on the non-blocking model, see the nonblocking_send function in the /usr/OV/prg_samples/ovsnmp_app/snmpdemo.c file that is shipped with the ovsnmp and the nvsnmp library.

OVsnmpDoRetry searches all active sessions for the calling process and, for each session, determines whether there are any pending requests that are due to be retransmitted. If there are, OVsnmpDoRetry sends the request to the specified destination and increments the number of times the request has been sent.

If the number of transmissions for a request is equal to the retries parameter coded on the OVsnmpOpen routine, the function specified by the callback parameter supplied to OVsnmpOpen is called with the command parameter set to SNMP_ERR_NO_RESPONSE.

Return Values

OVsnmpDoRetry returns NULL.

Error Codes

There are no errors returned by the OVsnmpDoRetry routine.

Libraries

When you are compiling a program that uses OVsnmpDoRetry, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpGetRetryInfo(3)” on page 499.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpRead(3)” on page 510.
- See “OVsnmpOpen(3)” on page 507.

OVsnmpErrString(3)

Purpose

Returns SNMP specific error strings

Syntax

```
#include <OV/OVsnmp.h>

char *OVsnmpErrString (int error)
```

Description

OVsnmpErrString returns a textual string that provides information about the error specified in the error parameter. If the error number is out of range, the string `Unknown Error` is returned.

Return Values

OVsnmpErrString returns a pointer to a static character string. This string is read only.

Libraries

When compiling a program that uses OVsnmpErrString, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpClose(3)” on page 440.

OVsnmpFixPdu(3)

Purpose

Deletes a variable with an error from an SNMP PDU

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpPdu *OVsnmpFixPdu(struct OVsnmpPdu *pdu, int type)
```

Description

OVsnmpFixPdu deletes a variable with an error from a response PDU and creates a new OVsnmpPdu structure. The OVsnmpPdu data structure that is returned will be of the type specified in the type parameter and will contain all of the variables in the response except the one that was in error. The error variable is determined by examining the error_index and error_status variables in the response PDU.

OVsnmpFixPdu should be called when a response is received and the error_status variable is not equal to SNMP_ERR_NOERROR.

OVsnmpFixPdu routine does not distinguish the type of error that is indicated by error_status, but only determines that one is indicated. If the calling process wants to examine the type of error that occurred, it must do so before calling OVsnmpFixPdu because the error status and error index are cleared.

On successful return from OVsnmpFixPdu, the input PDU will always be freed. The memory it referenced should not be used again.

Parameters

pdu
Specifies a pointer to an OVsnmpPdu data structure that contains the response.

type
Specifies the type of the new PDU command. This type must be one of the following values:

- GET_REQ_MSG
- GETNEXT_REQ_MSG
- SET_REQ_MSG
- TRAP_REQ_MSG

Return Values

If successful, OVsnmpFixPdu returns a pointer to a new OVsnmpPdu structure that can be used in a call to OVsnmpSend. If unsuccessful, it returns NULL.

Note: If successful, the memory associated with the input PDU is freed, so this memory must be dynamic. For a further discussion of dynamic memory, see memory rules on page 497.

Error Codes

Upon failure, OVsnmpFixPdu returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_ERR_BAD_PDU_TYPE] The PDU type for new PDU is not valid.

[SNMP_ERR_BAD_PDU] The input PDU was not a response.

[SNMP_SYSERR_MALLOC] Could not malloc space.

[SNMP_ERR_NO_VARS] There are no non-error variables left in the input PDU.

[SNMP_ERR_NO_ERRS] There is no error in the input PDU.

Libraries

When compiling a program that uses OVsnmpFixPdu, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpCreatePdu(3)” on page 490.
- See OVsnmpAddNullVarBind in man page “OVsnmpAddVarBind(3)” on page 435.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpBlockingSend(3)” on page 437.
- See “OVsnmpRecv(3)” on page 512.
- See “OVsnmpRead(3)” on page 510.

OVsnmpFreePdu(3)

Purpose

Frees all memory associated with the specified PDU

Syntax

```
#include <OV/OVsnmp.h>

void OVsnmpFreePdu(OVsnmpPdu *pdu)
```

Description

The OVsnmpFreePdu routine frees all memory associated with the specified PDU and the variables it contains. This memory must be dynamic; it must have been obtained by calls to one of the following routines or by direct calls to malloc:

OVsnmpCreatePdu
OVsnmpAddVarNullBind
OVsnmpAddVarTypedBind
OVsnmpFixPdu
OVsnmpRecv
OVsnmpRead

Many NetView for AIX SNMP API routines will free OVsnmpPdu data structures when sending data. However, if the FREE_PDU bit is not set in the session_flags variable of the session on which the PDU was sent, the OVsnmpPdu data structure will NOT be freed. In this case it is the caller's responsibility to free the OVsnmpPdu structure with a call to OVsnmpFreePdu.

Note: All memory associated with the PDU must be dynamic. This includes the OVsnmpPdu data structure as well as all SNMP variables referenced by the variables pointer in the OVsnmpPdu structure.

Parameters

pdu
Specifies a pointer to a dynamically allocated OVsnmpPdu structure. This structure can also contain pointers to dynamically allocated OVsnmpVarBind data structures.

Return Values

OVsnmpFreePdu does not return a value.

Libraries

When compiling a program that uses OVsnmpFreePdu, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpCreatePdu(3)” on page 490.
- See “OVsnmpAddVarBind(3)” on page 435.
- See “OVsnmpRecv(3)” on page 512.
- See “OVsnmpRead(3)” on page 510.
- See “OVsnmpFixPdu(3)” on page 495.

OVsnmpGetRetryInfo(3)

Purpose

Gets retransmission information about pending SNMP requests

Syntax

```
#include <OV/OVsnmp.h>
#include <sys/time.h>

int OVsnmpGetRetryInfo(fd_set *rfdsetp, struct timeval *tvp)
```

Description

The OVsnmpGetRetryInfo routine is intended to be used with select and OVsnmpDoRetry when you are using the ovsnmp or nvsnmp library in a nonblocking manner. The calling process uses OVsnmpGetRetryInfo to get retransmission information about pending SNMP requests.

OVsnmpGetRetryInfo fills in the timeval structure pointed to by the tvp parameter with the time the next pending SNMP request should be retransmitted. The fd_set data structure pointed to by the rfdsetp parameter is also assigned values that correspond to all SNMP-related file descriptors, so that the data can be used as the readfds parameter to select.

You can look at the example programs in the /usr/OV/prog_samples/ovsnmp_app/snmpdemo.c and /usr/OV/prog_samples/nvsnmp_app/filtertrap.c files for actual code that uses the OVsnmpGetRetryInfo call in the context of a non-blocking get operation.

Parameters

rfdsetp
Specifies that for each file descriptor with an pending SNMP request on it, the appropriate bit in the fd_set structure will be set to 1. The rfdsetp parameter is set such that it can be used as the readfds parameter to select for all active SNMP sessions.

Note: If you do not use the information in the fd_set data structure pointed to by the rfdsetp parameter, you might not receive arrival notification of an SNMP response during a subsequent call to select.

tvp
Specifies that if there are pending SNMP requests, as indicated by the return value from OVsnmpGetRetryInfo, the structure pointed to by tvp indicates the time the next retry should be done. This value should be used as the time-out parameter to select.

Note: If you do not use the information in the timeval data structure pointed to by the tvp parameter in a subsequent call to select, you might miss the retransmission time that you want for a pending SNMP request.

Return Values

OVsnmpGetRetryInfo returns the number of SNMP related file descriptors that have pending requests on them. This value can be used as the nfds parameter to select for all SNMP related file descriptors.

If the return value is greater than 0 (zero), the value pointed to by tvp will contain the interval until the next retry. If the return value is 0 (zero), the data structure pointed to by tvp will contain the value

MAX_ALARM as defined in the <sys/param.h> header file, which, if used as the time-out parameter to select, will cause the select call to block indefinitely.

Error Codes

There are no errors returned by OVsnmpGetRetryInfo.

Libraries

When compiling a program that uses OVsnmpGetRetryInfo, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpClose(3)” on page 440.
- See “OVsnmpDoRetry(3)” on page 492.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpRead(3)” on page 510.

OVsnmpIntro(5)

Purpose

Provides an introduction to the ovsnmp library

Description

The NetView for AIX SNMP Application Programming Interface (API) library is provided for applications that perform network management functions using the SNMP. The API sends and receives SNMP Protocol Data Units (PDUs) and performs ASN.1 encoding and decoding of the PDUs, location transparency, and minimal authentication.

Overview of Services

Each of the following sections provides more general information about the API. You should read the man pages listed under Related Information on page 506 with this introduction before developing an application.

The ovsnmp library is provided for the user who already has experience with the SNMP and application development using the SNMP.

Sending and Receiving SNMP PDUs

There are two ways to send and receive data. These ways provide access to the blocking and non-blocking behavior of the ovsnmp library.

Use OVsnmpSend to send SNMP PDUs in a nonblocking manner. You must register a callback function in the OVsnmpSession data structure that will be called when the response PDU is read or when a time-out occurs.

Use OVsnmpBlockingSend to send SNMP PDUs in a blocking manner (when you want to wait for the response before doing other processing). No callback function is needed when using OVsnmpBlockingSend.

Use OVsnmpRecv to receive a PDU on a single session. This call will block if there is no data available; therefore, use select or another function to verify that data is available. The received data is returned to the calling process. No callback function is used.

Use OVsnmpRead to receive data on several sessions at one time. Each active session with data available will result in a call to the specified callback function to make the data available to the user.

Retransmitting Lost PDUs

SNMP is implemented on UDP, which does not guarantee reliable packet delivery. Therefore, it is possible for PDUs to get lost on the network. The ovsnmp library enables you to retransmit lost PDUs.

Note: The library does not retransmit lost PDUs asynchronously. Use the OVsnmpGetRetryInfo and OVsnmpDoRetry routines to retransmit PDUs. To retransmit PDUs asynchronously, use the X11 extensions that are provided with the library.

The interval between retransmissions and the number of retransmissions are indicated by the appropriate parameters to OVsnmpOpen.

Location Transparency

The ovsnmp library provides functions that determine the location of an object manager when given the host name of the machine on which the object manager resides. This function can be used for proxy support. To access this function, register all proxy agents through the Options..SNMP Configuration pull-down menu when running NetView for AIX graphical user interface. Information about proxies is stored in the ovsnmp.conf file. See ovsnmp.conf(4).

You can specify a destination by setting the host name of the destination in the peer_hostname field in the session data structure.

Key Data Structures

This section discusses the primary data structures used in the ovsnmp library. These data structures are contained in the OVsnmpApi.h file.

OVsnmpSession Data Structure

The OVsnmpSession data structure is the primary data structure used by the ovsnmp library. Many of the functions in the library use the OVsnmpSession data structure as a parameter. You must allocate an OVsnmpSession structure with a call to OVsnmpOpen before using the other library calls.

The OVsnmpSession structure includes the following fields:

u_char *community
: Pointer to the community name of the peer with which you are communicating. If no community is specified, the default community, **public** is used. The memory associated with this field is dynamic. If community is not NULL on a call to OVsnmpClose, the memory pointed to by community will be freed. If the community name is NULL or has 0 (zero) length, or if there is no information about community name in the ovsnmp.conf file, the default community, **public**, is used.

u_int community_len
: The length of the community variable as returned by strlen.

int sock_fd
: The file descriptor used in sends and receives for this session.

u_short session_flags
: A bitmask for controlling the behavior of the session. The following flag is used:

  RECV_TRAPS
  : The session will receive traps. This flag is usually set by a call to OVsnmpRecvTraps, but it is permitted for an application to receive traps on a private port.

void (*callback)()
: Pointer to a function to be called when a PDU arrives for a session and the ovsnmp library is being used in a nonblocking manner. The function is called in response to OVsnmpRead.

  Note: If this variable is NULL, the library must not be used in a nonblocking manner. If the variable is NULL and a response PDU arrives, the next call to the library will result in an error.

void *callback_data
: Pointer to application-specific data that is passed to the callback function upon receipt of a PDU.

u_char *setCommunity
: Pointer to the community name for set requests on the peer with which you are communicating.

u_int setCommunity_len
: The length of the setCommunity variable as returned by strlen.

OVsnmpPdu Data Structure

The OVsnmpPdu data structure contains specific information needed to send an SNMP PDU. You should allocate and free these data structures with calls to OVsnmpCreatePdu and OVsnmpFreePdu respectively.

TheOVsnmpPdu structure includes the following fields:

ipaddr address	The IP address of the destination or responding host. When you are sending data using the default behavior, this field is ignored. When you are receiving data, the IP address will contain the address of the host that sent the PDU. The destination IP address is updated by the library. The application does not need to set this value. The IP address is ignored in a call to a send function.
int command	The SNMP-specific command for the PDU. This value must be one of the following: GET_REQ_MSG, GETNEXT_REQ_MSG, SET_REQ_MSG, TRP_REQ_MSG, GET_RSP_MSG.
int request_id	The request ID used when sending the PDU. This value is assigned by the OVsnmpSend calls. You should not modify this value.
int error_status	The error status for the PDU. When a PDU is received, this variable will indicate whether or not an error occurred.
int error_index	An index into the variable list. This value indicates the variable in the list that caused the error.
OVsnmpVarBind *variables	This is a pointer to a linked list of variables contained in this PDU. All memory referenced by this pointer is dynamic and will be freed by OVsnmpSend, OVsnmpFreePdu, OVsnmpFixPdu, or OVsnmpBlockingSend.

The following variables in the OVsnmpPdu structure are specific to traps.

ObjectID *enterprise	Pointer to the system object identifier used when sending and receiving traps. This variable references dynamic data. It will be freed by OVsnmpSend or OVsnmpFreePdu.
u_int enterprise_length	The length of enterprise as returned by strlen.
u_long agent_addr	The IP address of the agent that sent the trap.
int generic_type	SNMP-specific trap type information.
int specific_type	Enterprise-specific trap type.
u_long time	The system uptime for the agent that sent the trap.

OVsnmpVarBind Data Structure

The OVsnmpVarBind structure holds information about specific SNMP variables that a management station queries or sets and that an agent (object manager) returns. The structure itself and all data referenced by pointers in the OVsnmpVarBind structure are dynamically allocated by the ovsmnp library and can be freed by calls to OVsnmpSend, OVsnmpFreePdu, OVsnmpFixPdu, or OVsnmpBlockingSend.

The OVsnmpVarBind structure includes the following fields:

struct SNMPVarBind *next_variable	The next data structure in the NULL-terminated list of variables.

ObjectID *oid	Pointer to the object identifier for this variable. This is dynamic memory.
u_int oid_length	The number of elements or sub-identifiers in the object ID for this variable.
u_char type	The ASN.1 type of the variable.
OVsnmpVal val	The variable value.
u_int val_len	The number of elements in the value of the variable. This might not be the number of bytes in the value.

OVsnmpVal Data Structure

This is a union that contains pointers to the actual value of the variable. This union includes the following fields:

long *integer	If the type is integer, this variable contains a pointer to the value.
u_char *string	If the type is string, this variable contains a pointer to the value.
ObjectID *objid	If the type is objid, this variable contains a pointer to the value.

Configuration Data Structures

This section discusses the data structures used in the SNMP configuration routines, which are used to work with the ovsnmp configuration database. These data structures are contained in the OVsnmpConf.h file.

OVsnmpConfEntry Data Structure

The OVsnmpConfEntry data structure includes the following fields:

char *name	Pointer to the name of the target node. It can be a hostname, alias, IP address, or a proxy name.
char *community	The community name for SNMP requests.
char *setCommunity	The community name for SNMP set requests. If this parameter is not set, the value of community will be used.
char *proxy	The name of the proxy to use. If this parameter is set to SNMP_CONF_DONT_PROXY_STRING, no proxy will be assigned for target nodes whose names match the wildcard name, unless a proxy has been specifically configured for that target.
int timeout	The length of time, in tenths of seconds, to wait before retrying a request. This value must be greater than 0.
int retry	The number of times to try a request before giving up. This value must be greater than or equal to 0.
int pollInterval	The IP status polling interval, in seconds.
unsigned short remotePort	The SNMP port number on the target node.

OVsnmpConfDest Data Structure

The OVsnmpConfDest data structure includes the following fields:

short isProxied	1 if the target node is being proxied, otherwise 0.

unsigned long ipAddr — The IP address of the target node if it is not proxied; otherwise, the IP address of the proxy.

time_t ipAddrAge — The time at which the IP address was determined.

OVsnmpConfWcList Data Structure

The OVsnmpConfWcList data structure includes the following fields:

struct SNMPconfwclist *next — A pointer to the next element in the list.

OVsnmpConfEntry *confEntry — A pointer to an OVsnmpConfEntry data structure whose namefield contains a wildcard.

OVsnmpConfCntl Data Structure

The OVsnmpConfCntl data structure includes the following fields:

int ipAddrAge — The age limit, in seconds, for cached IP addresses.

cacheFlags_t cacheFlags

The caching features to use. This parameter can take on the following values:

- SNMP_CONF_CACHE_NONE, enable no caching.
- SNMP_CONF_CACHE_NOADDR, enable caching of SNMP configuration parameters is enabled but not caching of IP addresses.
- SNMP_CONF_CACHE_ALL, enable caching of both IP addresses and SNMP configuration parameters.

If caching is enabled, parameters are cached when OVsnmpConfResolveDest is first called for a target node. The parameters are returned on subsequent OVsnmpConfResolveDest calls.

compat3_2_t compatFlags

The level of shadowing to be performed. This parameter can take on the following values:

- SNMP_CONF_SHADOW_NONE, do no database shadowing.
- SNMP_CONF_SHADOW_32ONLY, shadow only those entries that are used by V2-bound applications.
- SNMP_CONF_SHADOW_ALL, shadow all entries.

For information on shadowing, see ovsnmp.conf.

Callback functions in nonblocking mode

The callback function that is registered in the OVsnmpSession data structure will be invoked as shown:

```
(void) callback (type, session, pdu, data)
```

The parameters to the callback function are described in the following list:

type — The type of command that caused the callback

session — The session on which the PDU was received

pdu: — The actual PDU that was received

data: — Application-specific data

Memory Management

All sending functions that have FREE_PDU set will free the PDU that was sent unless an error occurs. The calling process should not reference this data again.

Examples

Examples of SNMP calls are provided in the /usr/OV/prg_samples/ovsnmp_app directory.

Libraries

When compiling a program that uses the NetView for AIX SNMP API, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Implementation Specifics

The ovsnmp library is based on version 1.1 of the CMU library.

Related Information

- See “OVsnmpAddVarBind(3)” on page 435.
- See “OVsnmpDoRetry(3)” on page 492.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpBlockingSend(3)” on page 437.
- See “OVsnmpFixPdu(3)” on page 495.
- See “OVsnmpErrString(3)” on page 494.
- See “OVsnmpTrapOpen(3)” on page 517.
- See “OVsnmpRead(3)” on page 510.
- See “OVsnmpClose(3)” on page 440.
- See “OVsnmpFreePdu(3)” on page 497.
- See “OVsnmpRecv(3)” on page 512.
- See “OVsnmpCreatePdu(3)” on page 490.
- See “OVsnmpGetRetryInfo(3)” on page 499.
- See “OVsnmpSend(3)” on page 514.

OVsnmpOpen(3)

Purpose

Establishes an active SNMP session for communication with an SNMP agent

Related Functions

OVsnmpXOpen

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpSession *OVsnmpOpen (char *community, char *peername, int retries,
       int interval, u_short local_port, u_short remote_port,
       void (*callback)(), void *callback_data);

OVsnmpSession *OVsnmpXOpen (XtAppContext app_context, char *community,
       char *peername, int retries, int interval, short local_port,
       short remote_port, void (*callback)(), void *callback_data);
```

Description

The OVsnmpOpen and OVsnmpXOpen routines create an OVsnmpSession data structure that includes a UDP socket bound to the specified local port. The calling process must create an OVsnmpSession data structure prior to sending and receiving SNMP PDUs. The input parameters to OVsnmpOpen or OVsnmpXOpen will be used to control the behavior of the session. In many cases, the user might not want to specify values for all the parameters. Defaults are provided for the retries, interval, local_port and remote_port parameters.

The OVsnmpOpen and OVsnmpXOpen routines provide the same basic function; they both create an OVsnmpSession data structure and initialize it for use. However, the OVsnmpOpen function should be used when the calling process will manage retransmissions and receive data. The calling process is responsible for invoking OVsnmpGetRetryInfo to determine when the next retransmission of a pending request should be done. The calling process should use select to wait for a response to arrive. If the select call times out, the calling process should invoke OVsnmpDoRetry to retransmit the request. If data arrives before the select call times out, the calling process should invoke OVsnmpRead or OVsnmpRecv to receive the response.

You should use the OVsnmpXOpen call with X11. The calling process should use OVsnmpXOpen only if it is using XtMainLoop or an equivalent function to manage file I/O multiplexing. If the calling process is using X11 in an event driven manner, the OVsnmpSession data structure created should be used with the OVsnmpXSend function to send requests. See “OVsnmpSend(3)” on page 514 for details. In this case the calling process will manage retransmissions and receive the response. When the response arrives or a time-out occurs, the function specified by callback will be invoked by the ovsnmp library to process the response. The X11 interface to the ovsnmp library is simple, because it performs retransmissions for you. However, it is dependent on the X11 fileset. For more information about retransmissions, see Retransmitting Lost PDUs on page 501.

Parameters

community

Specifies a pointer to the community name that will be used in sending requests. If the community is NULL or of zero length, the community name will be determined by examining the ovsnmp.conf configuration file (see ovsnmp.conf(4)). If community is NULL or of zero length and there is no configuration information given in the ovsnmp.conf file, the default **public** will be used.

peername

Specifies a pointer to the destination for all requests sent on the session. This is specified using either an IP address or a host name (see hosts). It is an error if peername is NULL or of zero length.

This is the desired destination for all requests. If the specified destination is being proxied, requests might be sent to a different destination. See ovsnmp.conf(4) for more information.

retries

Specifies the number of times a pending request will be retransmitted before a time-out occurs. The default, **SNMP_USE_DEFAULT_RETRIES**, will cause the retry count to be retrieved from the ovsnmp.conf file. If no default retry value is listed in the ovsnmp.conf file and SNMP_USE_DEFAULT_RETRIES is used, pending requests will be retransmitted three times.

interval

Specifies the interval, in tenths of a second, between retransmissions. This value is exponentially increased between retransmissions. The default, **SNMP_USE_DEFAULT_INTERVAL**, will cause the retry count to be retrieved from the ovsnmp.conf file. If no default interval value is listed in the ovsnmp.conf file and SNMP_USE_DEFAULT_INTERVAL is used, the ovsnmp library will wait 5/10 of a second before doing the first retransmission.

local_port

Specifies the port to which the OVsnmpSession socket should be bound. The default, **SNMP_USE_DEFAULT_LOCAL_PORT**, will cause a port to be selected for you. If the caller supplies a port that is currently in use, an error occurs. If you need to know the port that was selected, use the getsockname call. For the calling process to receive traps on the OVsnmpSession socket rather than using the OVsnmpTrapOpen call, it will need to know its local port.

remote_port

Specifies the port on the machine specified by peername to which requests will be sent. The default, **SNMP_USE_DEFAULT_REMOTE_PORT**, will cause the port to be selected from the snmp/UDP entry in the services database. If there is no entry in the services database, **161** will be used as the remote port.

callback

Specifies a pointer to a function that will be invoked in response to an OVsnmpRead event when a PDU is received and you are using the ovsnmp library in a nonblocking manner. If this variable is NULL, the library must **not** be used in a nonblocking manner. If the callback variable is NULL and a response PDU arrives, the next call to the library will result in an error.

callback_data

Specifies a pointer to application specific data that will be passed to the callback function.

Return Values

If successful, OVsnmpOpen returns a pointer to a new OVsnmpSession structure. Any memory allocated for the OVsnmpSession structure must be freed by calling the OVsnmpClose routine.

If unsuccessful, OVsnmpOpen returns NULL.

Error Codes

OVsnmpOpen returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_SYSERR_SOCKET] The socket system call failed. The global variable errno contains the socket call specific error.

[SNMP_SYSERR_MALLOC] The malloc call failed. The global variable errno contains the malloc call specific error.

[SNMP_SYSERR_BIND] The bind call failed. The global variable errno contains the bind call specific error.

[SNMP_SYSERR_CONNECT] The connect call failed. The global variable errno contains the connect call specific error.

[SNMP_ERR_INVALID_HOST] The peername parameter is either a NULL pointer or of zero length. The global variable errno is not set.

[SNMP_ERR_GETHOSTBYNAME] The destination specified by the peername parameter is not an IP address or is not in the host's database. The global variable errno is not set.

Libraries

When compiling a program that uses OVsnmpOpen and OVsnmpXOpen, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpClose(3)” on page 440.
- See OVsnmpXClose in man page “OVsnmpClose(3)” on page 440.
- See “OVsnmpSend(3)” on page 514.
- See OVsnmpXSend in man page “OVsnmpSend(3)” on page 514.
- See “OVsnmpIntro(5)” on page 501.

OVsnmpRead(3)

Purpose

Receives SNMP messages on all active sessions

Syntax

```
#include  <OV/OVsnmp.h>

void OVsnmpRead(struct fd_set *rfdsetp)
```

Description

The OVsnmpRead routine is intended to be used with select-to-receive SNMP messages for all active sessions belonging to the calling process. Before calling the OVsnmpRead routine, you should first call select-to-wait for input to arrive on a session. The sock_fd variable in the OVsnmpSession data structure specifies the socket on which the response will be received. If there is no data available, OVsnmpRead will not take any action and will return.

The OVsnmpRead routine works on all calling process-sessions. If data is available on any session, it will be received.

If there is data available, the socket specified in the session is read to receive a PDU. The message is validated and decoded. The function specified by the callback variable to the OVsnmpOpen call is then invoked to process the inbound message. The specified callback function has the following prototype:

```
(void) callback (type, session_ptr, pdu, callback_data);
```

The callback-function parameters are described in the following list:

type
The type of PDU that caused the response. The type will be GET_REQ_MSG, GETNEXT_REQ_MSG, or SET_REQ_MSG.

Note: If a time-out occurs, the type parameter will be set to SNMP_ERR_NO_RESPONSE.

Additionally, the following exceptions are indicated by this parameter:

- [SNMP_ERR_NO_RESPONSE] indicates a timeout occurred. No response was received within the session timeout interval and retry count.
- [SNMP_SYSERR_LOSTCONN] indicates that the trapd process has stopped so the SNMP API can no longer receive traps on this session. This exception exists only for sessions created by OVsnmpTrapOpen. The calling process should close the trap session and no longer reference the session sock_fd in subsequent calls to select.

session_ptr
A pointer to the session that generated the request. This session should have been created by a call to OVsnmpOpen or OVsnmpTrapOpen.

pdu
A pointer to the OVsnmpPdu structure that contains the response information.

callback_data
A pointer to application-specific data registered for the session by OVsnmpOpen or OVsnmpXOpen. This is application-specific data unique to the session. This data is registered on a call to OVsnmpOpen.

The ovsnmp library will not free the response PDU that is delivered to the calling process. It is the responsibility of the calling process to free the PDU with a call to OVsnmpFreePdu when done processing the data.

Parameters

rfdsetp
A pointer to a structure that indicates which sessions have input available on the socket. This variable should have been initialized by a call to select before calling OVsnmpRead.

Libraries

When compiling a program that uses OVsnmpRecv, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpDoRetry(3)” on page 492.
- See “OVsnmpRecv(3)” on page 512.
- See “OVsnmpGetRetryInfo(3)” on page 499.
- See “OVsnmpOpen(3)” on page 507.

OVsnmpRecv(3)

Purpose

Receives an SNMP PDU for a specific session

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpPdu *OVsnmpRecv(OVsnmpSession *session)
```

Description

An SNMP PDU is received on the socket specified in the sock_fd variable in the session pointed to by the session parameter. The PDU is then parsed to insure its validity. The type and source address of the PDU are returned in the OVsnmpPdu structure with PDU-specific errors.

Error status is indicated by the value of the error_status variable in the OVsnmpPdu structure that is returned to the calling process. If the value of the error_status variable is SNMP_ERR_NOERROR, the returned PDU does not contain any errors. Otherwise, the variable in the returned PDU that has an error associated with it is identified by the error_index variable in the returned PDU.

If an error occurs, the value of the error_status variable can be passed to the OVsnmpErrString function, which will return a textual description of the error.

The calling process should have first determined that data is available on the socket by a call to select or by another method.

Note: If there is no data available on the socket, OVsnmpRecv will block until data arrives.

The OVsnmpPdu structure that is returned is created from dynamic memory. It should be freed by a call to OVsnmpFreePdu or OVsnmpFixPdu.

Parameters

session
: Pointer to a valid OVsnmpSession structure returned by a call to OVsnmpOpen. This structure contains the socket descriptor and other information needed to receive an SNMP PDU.

Return Values

If successful, OVsnmpRecv returns a pointer to the PDU that was received. If unsuccessful, it returns NULL.

Error Codes

OVsnmpRecv returns the error code value OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call. If an error occurs, the inbound message can be discarded. The following list describes the possible errors.

[SNMP_SYSERR_RECVFROM] A call to recvfrom failed.

[SNMP_SYSERR_MALLOC] A call to malloc failed.

[SNMP_ERR_BAD_SESSION]	The session parameter is not valid.
[SNMP_ERR_PARSE_ERR]	Could not parse message. Message is dropped.
[SNMP_SYSERR_LOSTCONN]	The trapd process has stopped and the SNMP API can no longer receive traps on this session. This error exists only for sessions created by OVsnmpTrapOpen. The calling process should close the trap session sock_fd in subsequent calls to select.

Libraries

When compiling a program that uses OVsnmpRead, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpRead(3)” on page 510.
- See “OVsnmpSend(3)” on page 514.
- See “OVsnmpFixPdu(3)” on page 495.

OVsnmpSend(3)

Purpose

Sends an SNMP PDU in nonblocking mode

Related Functions

OVsnmpXSend

Syntax

```
#include  <OV/OVsnmp.h>

int OVsnmpSend( OVsnmpSession *session, OVsnmpPdu *pdu);

int OVsnmpXSend( OVsnmpSession *session, OVsnmpPdu *pdu);
```

Description

The OVsnmpSend and OVsnmpXSend routines perform ASN.1 encoding on the PDU and send the PDU to the host specified for the session. The destination host is provided as a parameter to the OVsnmpOpen or OVsnmpXOpen calls.

Both the OVsnmpSend and OVsnmpXSend routines operate in a nonblocking manner. The specified PDU is encoded and sent without waiting for a response. You should use OVsnmpBlockingSend if you want to use the ovsnmp or nvsnmp library in a blocking manner.

The OVsnmpSend and the OVsnmpXSend routines perform the same basic services. The difference in the two functions is in the way retransmissions and receive processes are handled.

The OVsnmpSend function is intended for to be used with OVsnmpGetRetryInfo, OVsnmpDoRetry, OVsnmpRead, and select. This enables the calling process to monitor its file descriptors and manage retransmissions of pending requests.

Once the PDU passed to OVsnmpSend has been transmitted, the calling process should invoke OVsnmpGetRetryInfo to determine when pending requests should be retransmitted. The caller should then use select or another method to wait for the response to arrive. If the select call times out, the caller should invoke OVsnmpDoRetry to retransmit the request. If the request arrives before the time-out, the caller should invoke OVsnmpRead to receive the data and pass it to the specified callback function. For an example of this coding model, see the files in the /usr/OV/prg_samples/ovsnmp_app directory.

Note: If OVsnmpSend is successful or an SNMP_ERR_NO_RESPONSE error occurs, and the FREE_PDU bit in the session_flags is turned on (default case), the request pdu parameter is freed by OVsnmpSend. The memory associated with the pdu parameter should not be referenced again. However, if OVsnmpSend returns another error, it does not free the memory for the pdu parameter. If the FREE_PDU bit in the session_flags has been explicitly turned off by the calling process, the memory associated with the pdu parameter is never freed by OVsnmpSend. The calling process must free the pdu with a call to OVsnmpFreePdu. If this is not done, the calling process will consume unneccessary amounts of memory.

Use the OVsnmpXSend routine when the calling process is using XToolkit and X is managing the calling processes' file descriptors. The calling processes should use the OVsnmpXSend routine only if they are using XtMainLoop or an equivalent function to manage file I/O multiplexing. Use the OVsnmpXSend

routine with the OVsnmpXOpen and OVsnmpXClose routines. For more information, see the X documentation.

To use the OVsnmpXSend routine, the calling process will register an application context with the OVsnmpXOpen routine. The xt application context is returned by a call to XtCreateApplicationContext.

Once the calling process has completed these tasks, the retransmissions and the receipt of the response is handled by the ovsnmp library. The callback function that was registered with the OVsnmpXOpen call will be invoked by the ovsnmp library when a response arrives or when the request times out. The callback occurs in response to an OVsnmpRead event.

Parameters

session
A pointer to an OVsnmpSession structure returned by a call to OVsnmpOpen or OVsnmpXOpen. The OVsnmpSession structure determines the destination, retry information, callback function pointer, and optionally, a community name, authenticator function pointer, and user data to be used in transmitting the SNMP PDU specified in the pdu parameter.

pdu
A pointer to an OVsnmpPdu structure returned by a call to OVsnmpCreatePdu. The pdu structure contains the PDU type and a pointer to the OVsnmpVarBind list of the variables that are being sent. Variables are added to an OVsnmpPdu structure with either the OVsnmpAddNullVarBind call or the OVsnmpAddTypedVarBind call. Generally, the OVsnmpAddNullVarBind function will be used to add variables that the caller wants to retrieve and the OVsnmpAddTypedVarBind function will be used to add variables the caller wants to set.

Return Values

If successful, OVsnmpSend returns the request ID for the PDU. If unsuccessful, it returns –1 (negative one).

Error Codes

OVsnmpSend returns the error code value of OVsnmpErrno. If one of the SNMP_SYSERR_* values are returned in OVsnmpErrno, the external variable errno contains the error value returned by the failed system call.

[SNMP_ERR_BAD_SESSION]	The session parameter is not valid.
[SNMP_ERR_PDU_BUILD]	Could not build the ASN.1 encoded PDU.
[SNMP_ERR_GETHOSTBYNAME]	A call to gethostbyname failed.
[SNMP_SYSERR_SENDTO]	A call to sendto failed.
[SNMP_SYSERR_MALLOC]	A call to malloc failed.

Libraries

When compiling a program that uses OVsnmpSend, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpIntro(5)” on page 501.
- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpClose(3)” on page 440.
- See “OVsnmpDoRetry(3)” on page 492.
- See “OVsnmpGetRetryInfo(3)” on page 499.
- See “OVsnmpRead(3)” on page 510.
- See “OVsnmpRecv(3)” on page 512.
- See “OVsnmpCreatePdu(3)” on page 490.
- See “OVsnmpAddVarBind(3)” on page 435.

OVsnmpTrapOpen(3)

Purpose

Connects to the trapd daemon and sets up to receive unfiltered traps

Related Functions

OVsnmpXTrapOpen

Syntax

```
#include  <OV/OVsnmp.h>

OVsnmpSession *OVsnmpTrapOpen (void (*callback)(), void *callback_data);

#include  <OV/OVsnmp.h>
#include  <OV/OVsnmpXfns..h>

OVsnmpSession *OVsnmpXTrapOpen (XtAppContext context, void (*callback)(),
       void *callback_data);
```

Dependencies

OVsnmpTrapOpen is dependent on the trapd process. If the trapd process is not running, OVsnmpTrapOpen will fail.

Description

OVsnmpTrapOpen and OVsnmpXTrapOpen create an active SNMP session that is used explicitly for receiving traps. You cannot send any data and can only receive traps on this session. The new session communicates with the trapd process and receives all traps that are sent to the trapd process. No scoping or filtering is provided by OVsnmpTrapOpen. If you want filtering services, refer to "OVsnmpTrapOpen(3)." The new session can be used with OVsnmpRead or OVsnmpRecv to receive traps.

If the callback parameter is non-NULL and OVsnmpRead is used to receive a trap, the function specified by the callback parameter will be invoked to process the inbound trap. It is an error to use OVsnmpRead with a NULL callback function.

If the calling process will not use callback functions, the OVsnmpRecv must be used to receive traps. OVsnmpRecv returns a pointer to the trap PDU that was received. It does not use the callback function.

Use the OVsnmpXTrapOpen call with X11. The calling process should use OVsnmpXTrapOpen only if it is using XtMainLoop, or an equivalent function, to manage file I/O multiplexing. OVsnmpXTrapOpen will install the newly created file descriptor as an input device that X11 will use to detect data. When a trap arrives on the new file descriptor, the callback function supplied in the callback parameter will be invoked by the ovsnmp library. The calling process does not perform additional processing.

Parameters

callback
A pointer to the routine that will be invoked to process inbound traps if the OVsnmpRead function is used to receive the trap. In order for the calling process to use callback procedures, this parameter must point to a valid function.

callback_data
A pointer to application specific data that will be passed to the callback function when it is invoked. The ovsnmp library does not perform any action on this data.

Keyword Syntax

! NOT (logical negation)

&& AND (logical 'and')

|| OR (logical 'or')

The following list describes the keywords in the syntax used to define filters:

CLASS=value
SNMP enterprise match on enterprise ID. Value is given in dot notation (1.2.3.4.55, for example).

IP_ADDR=value
SNMP agent-addr match on IP address. Value is given in dot notation (192.155.13.57, for example). Registration for an IP_ADDR permits receipt of agent-generated traps, as well as internal events related to that IP_ADDR.

LOGGED_TIME <= time_string
Time that was logged before the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

LOGGED_TIME >= time_string
Time that was logged after the time in time_string, where time_string has the form dd:mm:yy:hh:mm:ss (24 hour clock, GMT)

PRESENT = SNMP_TRAP
Presence of SNMP Trap

SNMP_TRAP=value
Match on SNMP Generic Trap Type, where the Generic Type is an integer

SNMP_SPECIFIC=value
Match on SNMP Specific Trap Type, where the Specific Type is an integer

TIME_PERIOD=time_constant
Relative time period (integer seconds) for frequency filters

THRESHOLD <= frequency
Number of event occurrences is less than or equal to frequency (integer) during TIME_PERIOD

THRESHOLD >= frequency
Number of event occurrences is greater than or equal to frequency (integer) during TIME_PERIOD

Note: When included in an expression for nvSnmpTrapOpenFilter, the keywords THRESHOLD and TIME_PERIOD must be ANDed (never ORed) and grouped within parentheses as in the following example:

```
filter = PRESENT=SNMP_TRAP && (THRESHOLD <= 5 && TIME_PERIOD = 30)
```

Return Values

If successful, OVsnmpTrapOpen returns a pointer to a new OVsnmpSession structure. Memory allocated for the OVsnmpSession structure must be freed by OVsnmpClose.

If unsuccessful, OVsnmpTrapOpen returns NULL.

Note: The trapd process might stop after OVsnmpTrapOpen or OVsnmpXTrapOpen returns successfully. Therefore, the callback function provided by the calling process should handle the SNMP_SYSERR_LOSTCONN exception condition indicated in the type parameter of the callback function.

Error Codes

OVsnmpTrapOpen returns the error code value of OVsnmpErrno. If one of the SNMP_SYSERR_* values is found, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

[SNMP_SYSERR_SOCKET]	A call to socket failed.
[SNMP_SYSERR_MALLOC]	A call to malloc failed.
[SNMP_SYSERR_BIND]	A call to bind failed.

Libraries

When compiling a program that uses OVsnmpTrapOpen, you need to link to the following libraries:

- /usr/OV/lib/libovsnmp.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Note: If your program also uses nvSnmp calls, replace /usr/OV/lib/libovsnmp.a with /usr/OV/lib/libnvsnmp.a.

Related Information

- See “OVsnmpOpen(3)” on page 507.
- See “OVsnmpClose(3)” on page 440.
- See trapd(8)

OVsPMD_API(3)

Purpose

Describes routines for well-behaved daemon processes in the NetView for AIX program

Related Functions

OVsInit
OVsInitComplete
OVsReceive
OVsDone

Syntax

```
#include <OV/OVsPMD.h>
int OVsInit(int *sp);
int OVsInitComplete(OVsCodeType code, char *message);
int OVsReceive(OVsPMDCommand *command);
int OVsDone(char *message);
```

Description

The described routines are used by object managers (agents) that must run as background processes in the NetView for AIX program in order to be managed by the ovspmd daemon, the process management daemon. An object manager that uses these routines as described is considered well-behaved and should be configured OVs_WELL_BEHAVED in its LRF. See lrf.

The ovspmd daemon starts and stops all object managers that run as daemons. A well-behaved object manager uses the OVsPMD API to communicate with the ovspmd daemon so that the ovspmd daemon can manage it.

The OVsPMD API is not available on clients in a client/server environment.

A well-behaved object manager interacts with the ovspmd daemon as follows:

1. The object manager must not go into the background on its own. The ovspmd daemon starts each object manager in the background, as its child. If the object manager process forks and the parent exits, the ovspmd daemon no longer has access to the process ID of its child process, and can no longer manage it.
2. When initializing, the object manager must call OVsInit, which returns, in the location pointed to by sp, a file descriptor for interprocess communication with the ovspmd daemon. If this call fails, communication with the ovspmd daemon will be impossible.
3. After initializing, (whether successfully or not), the object manager must call OVsInitComplete to notify the ovspmd daemon. The code parameter is OVS_RSP_SUCCESS, if initialization succeeded, or OVS_RSP_FAILURE if it failed.

   The ovspmd daemon will wait for the object manager to make the OVsInitComplete call before starting other object managers that depend on it. The code and message parameters are sent to the ovstart command. In the case of OVS_RSP_SUCCESS, the message is sent out by the ovstart command in verbose mode only.

   If initialization failed, the ovspmd daemon expects the object manager to call OVsInitComplete with code set to OVS_RSP_FAILURE, and exit immediately. It should not call the OVsDone routine. The

message parameter is sent to the user by the ovstart command. This message should tell the user why initialization failed, and might also provide a solution. The ovspmd daemon will not start any other object managers that depend on an object manager that has failed to initialize.

After the object manager has called the OVsInitComplete routine with the code parameter set to OVS_RSP_SUCCESS, the message can be updated by subsequent calls to the OVsInitComplete routine with the code parameter set to OVS_RSP_SUCCESS and the message parameter set to the new message. The ovstatus routine will send the latest message that the ovspmd daemon has received. The object manager may continue to make calls to the OVsInitComplete routine until it calls the OVsDone routine.

4. It is assumed that the object manager is organized as a loop around a select call, waiting for input from applications or from the managed object. The object manager should use select for reading on the file descriptor returned by the OVsInit routine.
5. When select indicates that the file descriptor is readable, the object manager must call OVsReceive to receive a command from ovspmd in command. The object manager must take appropriate action, based on the value of command.code received. The implemented command code is OVS_CMD_EXIT; the correct response to this command code is to immediately clean up, call OVsDone, and exit.
6. When the object manager exits, it must inform the ovspmd daemon. If the object manager exits in response to OVS_CMD_EXIT, the OVsDone call notifies the ovspmd daemon that the object manager is exiting and that it should not be sent to SIGTERM or SIGKILL. If the object manager exits spontaneously, without having received OVS_CMD_EXIT, it should call OVsDone and use the message parameter to notify ovspmd why it exited. The ovspmd daemon will log the exit of the object manager, including the message it sent.

Return Values

If successful, OVsInit returns 0 (zero) and OVsInitComplete, OVsReceive, and OVsDone return greater than 0 (zero).

If unsuccessful, these routines return –1 (negative one). If a system call on which they depend failed, errno will be set to indicate the problem. Failure of any of these routines indicates that the daemon has lost contact with ovspmd. A well-behaved daemon should exit if any of these routines fail.

Libraries

When compiling a program that uses the OVsPMD_API, you need to link to the following libraries:

- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovaddobj.
- See ovdelobj.
- See ovspmd.
- See ovstart.
- See ovstatus.
- See ovstop.
- See “OVuTL(3)” on page 522.
- See lrf.

OVuTL(3)

Purpose

Facilitates NetView for AIX logging and tracing

Related Functions

OVuTLInit
OVuLog
OVuTrace

Syntax

```
#include  <OV/OVuTL.h>

int OVuTLInit(int argc,
              char *argv[]);

int OVuLog(unsigned int class, char *msgcat, int setnum, int msgnum, char *fmt);

int OVuTrace(unsigned int kind, char *msgcat, int setnum, int msgnum, char *fmt);
```

Description

OVuLog and OVuTrace enable programs to issue logging and tracing messages through the nettl tracing and logging facility used by the NetView for AIX server. This facility is controlled using the nettl command and its output is viewed using the netfmt command.

Logging: Logging is typically run at all times.

The user who sees an error message at the time the error occurs may be neither authorized nor knowledgeable enough to fix the problem. Therefore, both failures and normal transactions are logged, so that the administrator can access the history of subsystem operation.

Log messages should be interpretable by the administrator or user.

Tracing: Tracing is typically run only when a problem has occurred, and it is necessary to reconstruct in detail the sequence of events that caused the problem.

Tracing messages may not be intended to be interpreted by an unaided administrator or user. There are three general contexts in which tracing is used:

- Tracing is frequently used as an aid to code development by both the developers of the code that actually performs the tracing, such as procedure entry/exit tracing and developers of applications that interact with tracing, such as API tracing.
- Tracing is used by users and administrators for troubleshooting system or network problems. For example, protocol tracing can be used for this purpose.
- Tracing is used by the field and support personnel, in conjunction with the lab personnel and the customer, for diagnosing customer problems. All supported kinds of tracing can be used for this purpose.

Functions: OVuTLInit initializes the software and hostname fields in the logging and tracing output. See the output examples under Examples on page 525. It should be called in main before OVuLog or

OVuTrace is called. Otherwise, the software field will not be set, making it difficult for the reader of the tracing or logging output to determine which program issued the trace or log message.

OVuLog logs a message using the nettl facility. See nettl for more information. The same message is also sent as a logging-trace message, if tracing is enabled for the OVEXTERNAL subsystem and the LOGGING_TRACE_BIT Trace Kind.

OVuTrace sends a trace message using the nettl facility.

Parameters

argc Specifies the argument count. This should be the same argc passed as a parameter to main.

argv Specifies the argument vector. This should be the same argv passed as a parameter to main. Argv is not modified and is used only to set the software field in the log or trace output.

class Specifies the log Class (OVuLog). The permitted values of class are specified in the header files OV/OVuTL.h and subsys_id.h as follows:

DISASTER DISASTER log messages refer to failures that compromise the entire subsystem, and make further normal operation impossible.

DISASTER class messages are logged to the console as well as to the log file, if console logging is enabled. Note that the message logged to the console contains only a timestamp, the log class, and the software and hostname fields. This information can be used to find the rest of the message in the log file.

ERROR ERROR log messages refer to failures affecting only the particular transaction.

WARNING WARNING log messages refer to events that are probably failures in cases where the subsystem will attempt to complete the operation anyway. Warnings should generally be issued only when they suggest some action the administrator could take to reduce the likelihood of a future Error or Disaster.

INFORMATIVE INFORMATIVE log messages report significant but normal events. The information provided should be sufficient for the administrator to reconstruct a history of transactions that have taken place.

kind Trace Kind (OVuTrace). The permitted values of kind are specified in the header files OV/OVuTL.h and subsys_id.h as follows:

HDR_IN_BIT Inbound header tracing

HDR_OUT_BIT Outbound header tracing

PDU_IN_BIT Inbound Protocol Data Unit tracing

PDU_OUT_BIT Outbound Protocol Data Unit tracing

These trace kinds are used for network protocol tracing.

PROCEDURE_TRACE_BIT

Procedure entry/exit trace

This trace kind is used to trace function entry and exit. On entry, all input parameter values should be traced. On exit,

	the return value and the values of output parameters should be traced.
STATE_TRACE_BIT	State machine tracing If the code is organized as a state machine, this trace kind can be used to trace state transitions.
ERROR_TRACE_BIT	Error tracing Use of this trace kind should be limited to errors that are NOT logged. For example, system errors from which the code automatically recovers should be traced rather than logged.
LOGGING_TRACE_BIT	Log call tracing Logging tracing is used to record the fact that a log message was issued. It is used by the troubleshooter to synchronize log messages with tracing output, because tracing output goes to a different file than logging output. OVuLog automatically generates a logging trace, in addition to the log message, if logging tracing is enabled for the subsystem.
LOOP_BACK_BIT	Loopback
PTOP_BIT	Point to point LOOP_BACK_BIT and PTOP_BIT are reserved for use by kernel networking and should not be used by NetView for AIX applications.
SUBSYSTEM_BITS	Subsystem specific trace kinds Subsystems can define their own trace kinds. There is one defined specifically for NetView for AIX in the OV/OVuTL.h header file.
API_TRACE_BIT	API tracing is used by developers to determine whether parameters to the NetView for AIX APIs have been formatted correctly. This should be used exactly as procedure entry/exit tracing is used, except that only exported routines are traced and these routines are traced only on entry.

msgcat A pointer to the name of the message catalog file containing the format string, as used by catopen.

The message catalog is opened by netfmt, not by the code calling the OVuTL routines, when formatting the messages. Only the format string is searched for in the message catalog. It is the responsibility of the calling program to localize other arguments.

setnum Number of the message, set within the message catalog containing the format string, which is used by catgets.

msgnum Index of the format string in the message set, which is used by catgets.

fmt A pointer to the default format string in printf syntax. This format string is used as in catgets if the message catalog cannot be accessed.

In addition to the % conversion specifiers understood by printf, %m will be replaced by the string that is returned by strerror, which is called with the current value of errno as in syslog.

arg Zero or more arguments are formatted according to the format string, as in printf.

Return Values

If successful, OVuTLInit returns 1. If it is unable to set the host name or software fields, OVuTLInit returns 0 (zero).

If successful, OVuLog and OVuTrace return 1.

If logging is not enabled for the OVEXTERNAL subsystem or for the specified Log Class, OVuLog returns –1 (negative one). If tracing is not enabled for the OVEXTERNAL subsystem or for the specified trace kind, OVuTrace returns –1 (negative one). In these cases, every effort is made to minimize processing. See nettl.

If a syntax error is detected in fmt, an internal buffer overflow occurs, or an error occurs in passing the message to the nettl subsystem, OVuLog and OVuTrace return 0 (zero).

Error Codes

The OVEXTERNAL subformatter, part of netfmt, detects buffer overflow conditions and certain printf format errors, and outputs an error message instead of, or in addition to, the text of the failed log or trace message.

Examples

- Logging

  In the following example, 71 is the index of the format in message set number 1 of the toasterM message catalog. Note the explicit return code processing.

```
int r;

r = OVuLog(INFORMATIVE, "toasterM", 1, 71,
"  Object Instance      : %d\n\
Brownness control adjusted to %4.1f%%\n",
instance, value);
if (r > 0)
{
       /* logging succeeded */
}
else if (r < 0)
{
       /*
       **  nettl not running, or
       **  logging for OVEXTERNAL INFORMATIVE messages not enabled
       */
}
else if (r == 0)
{
       /* OVuLog failed: error processing */
}
```

- Logging of INFORMATIVE messages in the OVEXTERNAL subsystem is enabled with the following command:

```
/usr/OV/bin/nettl -entity OVEXTERNAL -log i w e d
```

The output is read using a command similar to the following command:

```
/usr/OV/bin/netfmt -t 1 -f /usr/OV/log/nettl.LOG00
```

The output is shown in Table 16:

Table 16. Output of logging INFORMATIVE messages in the OVEXTERNAL subsystem

Timestamp	: Tue Apr 02 1991 12:32:58.818232		
Process ID	: 940	Subsystem	: OVEXTERNAL
User ID (UID)	: 214	Log Class	: INFORMATIVE
Device ID	: –1	Path ID	: –1
Connection ID	: –1	Log Instance	: 0
Software	: /usr/OV/bin/toasterM		
Hostname	: breadbrain.toastmaster.com		
Object Instance	: 47		
Brownness control adjusted to 85.0%.			

The previous OVuLog call generates a logging trace message, if logging tracing is enabled for the OVEXTERNAL subsystem.

Tracing: The following is a procedure entry trace using OVuTrace, and the output it generates. Note the streamlined return code processing. In most cases, the calling program will not need to handle success differently from not enabled.

```
if (!OVuTrace(PROCEDURE_TRACE_BIT, msgcat, 1, 114,
"  Entering function   : AdjustBrownness()\n\
  File                : %s\n\
  Line                : %d\n\
  Parameter values    :\n\
    instance    =    %d\n\
    value       =    %10.4e\n",
__FILE__, __LINE__, instance, value))
{
    /* OVuTrace returned 0: error processing */
}
```

Procedure tracing in the OVEXTERNAL subsystem is enabled with the command:

```
/etc/nettl -entity OVEXTERNAL -traceon proc -file
/usr/OV/log/nettl
```

The output is read using a command similar to the following command:

```
/etc/netfmt -t 1 -f
/usr/OV/log/nettl.TRC0
```

The output is shown in Table 17:

Table 17. Output of Tracing in OVEXTERNAL Subsystem

Timestamp	: Tue Apr 02 1991 12:32:58.815152		
Process ID	: 940	Subsystem	: OVEXTERNAL
User ID (UID)	: 214	Trace Kind	: Proc. entry/exit
Device ID	: –1	Path ID	: –1
Connection ID	: –1		

Software	: /usr/OV/bin/toasterM
Node	: breadbrain.toastmaster.com

Entering function	: AdjustBrowness()
File	: actions.c
Line	: 456
Parameter values	:
instance = 47	
value = 8.5000e-02	

Libraries

When compiling a program that uses OVuTL, you need to link to the following libraries:

- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See netfmt.
- See nettl.

OVwAckMapClose(3)

Purpose

Acknowledges a map close event

Syntax

```
#include <OV/ovw.h>

int OVwAckMapClose(OVwMapInfo *map, time_t close_time);
```

Description

OVwAckMapClose must be called in response to the ovwMapClose event. It informs the NetView for AIX program that the application has received the ovwMapClose event and has completed any changes needed before the NetView for AIX program closes the map.

Note: Failure to call OVwAckMapClose in response to the ovwMapClose event will cause the graphical interface to wait for several seconds, as specified by the closeTimeout X resource for ovw, before closing the map.

Parameters

map
Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo, saved from the ovwMapOpen event using OVwCopyMapInfo, or can be the same map parameter returned by the OVwMapCloseCB callback.

close_time
Specifies the proposed closing time for the map. If the proposed closing time passed to the OVwMapCloseCB callback routine is acceptable, the close-time parameter can be set to the proposed closing time or a default value of 0 (zero). Alternately, the close-time parameter can be set to an earlier time needed by the application, so it can correctly synchronize with the map when it is reopened. The final last closing time parameter for the map will be set to the earliest closing time parameter specified by any application.

Return Values

If successful, OVwAckMapClose returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwAckMapClose sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following is a sample callback routine for an ovwMapClose event:

```
void
mapCloseProc(void *user_data, OVwEventType type,
        OVwMapInfo *map, time_t closing_time)
{
        time_t new_close_time = (time_t) 0;

    /*
    * If necessary, compute an earlier new_close_time
    * based on what needs to be done next time the map
    * is opened.
    */
    OVwAckMapClose(map, new_close_time);
}
```

Implementation Specifics

OVwAckMapClose supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAckMapClose, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwMapCloseCB(3)” on page 759.
- See “OVwApiIntro(5)” on page 560.

OVwAckUserSubmapCreate(3)

Purpose

Acknowledges a user, submap-create event

Syntax

```
#include <OV/ovw.h>

int OVwAckUserSubmapCreate(OVwMapInfo *map, OVwSubmapId submapId,
       OVwSymbolId symbolId);
```

Description

OVwAckUserSubmapCreate must be called in response to an ovwUserSubmapCreate event (see “OVwUserSubmapCreateCB(3)” on page 831). This routine acknowledges the ovwUserSubmapCreate event and indicates whether the application has created a submap in response to the user's attempt to open a symbol, through the graphical interface, that does not have a child submap.

If it is appropriate for the application to create a submap for the symbol the user is trying to open, it can call OVwCreateSubmap to create a new submap and pass the submap ID of the new submap as the submap_id parameter of the OVwAckUserSubmapCreate call. The graphical interface will then display the specified submap to the user. If the application does not create a submap for the symbol, ovwNullSubmapId can be returned. In this case, the graphical interface will prompt the user to create a submap for the object associated with the symbol.

Parameters

map
Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submapId
Specifies the submap ID of the submap created by the application to display or ovwNullSubmapId.

symbolId
Specifies the symbol ID of the symbol the user is trying to open. This is available from the OVwSymbolInfo structure passed by the OVwUserSubmapCreateCB callback.

Return Values

If successful, OVwAckUserSubmapCreate returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwAckUserSubmapCreate sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap specified by submapId does not exist on the open map.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Examples

The following example illustrates the use of OVwAckUserSubmapCreate in an OVwUserSubmapCreateCB callback:

```
void userSubmapCreateCB(void *userData, OVwEventType type,
        OVwMapInfo *map, OVwSymbolInfo *symbol,
        OVwSubmapInfo *submap)
{
        OVwSubmapId submap_id = ovwNullSubmapId;

        /*
         * Check whether it is appropriate for the application
         * to create a submap.  If so, create submap and set
         * submap_id.
         */

        OVwAckUserSubmapCreate(map, submap_id, symbol→symbol_id);
}
```

Implementation Specifics

OVwAckUserSubmapCreate supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAckUserSubmapCreate, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwInit(3)” on page 741.
- See “OVwUserSubmapCreateCB(3)” on page 831.
- See “OVwApiIntro(5)” on page 560.

OVwAddActionCallback(3)

Purpose

Registers a callback for a registered action

Related Functions

OvwRemoveActionCallback

Syntax

```
#include <OV/ovw.h>

void (*OVwActionCallbackProc) (void *userData, char *actionId,
       char *menuItemID, OVwObjectIdList *selections,
       int argc, char **argv, OVwMapInfo *map, OVwSubmapId submap);

int OVwAddActionCallback(char *actionId,
       OVwActionCallbackProc callbackProc, void *userData);

int OVwRemoveActionCallback(char *actionId);
```

Description

OVwAddActionCallback associates an application procedure with an action registered in the application registration file. The procedure will be started when the specified action is triggered by using a menu item or an executable symbol.

OVwRemoveActionCallback removes previously registered callbacks for the specified action.

Parameters

actionId
Specifies a pointer to an action name as declared in the application's registration file. If actionId is NULL, the callback is invoked for any action notification that does not have a corresponding registered callback. This enables you to receive all action callbacks by one procedure.

argc
Specifies the count of the number of arguments contained in the CallbackArgs statement of the action registration.

argv
Specifies an argument vector containing any arguments specified in the CallbackArgs statement of the action registration. The first argument is argv[0]

callbackProc
Specifies a procedure to be called when a user activates the specified menu item.

map
Specifies a pointer to a MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

menuItemID
If the action was triggered by the selection of a menu item, this parameter will be a pointer to the label of that menu item. If the action is started by an executable symbol, the menuItemID parameter will be NULL.

selections
Specifies a pointer to a list of the objects that were in the selection list when the action was triggered. When the action was triggered from an object menu, the selections parameter contains the name of the object from whose menu the action was invoked.

submap
Specifies the ID for the submap where the action was triggered.

userData
Specifies a pointer to the application-specific data required for the callback procedure.

Return Values

If successful, OVwAddActionCallback and OVwRemoveActionCallback return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddActionCallback and OVwRemoveActionCallback set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_FOUND] The action specified by actionId is not registered by the application.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been started with OVwInit.

[OVw_ACTION_NOT_APPLICABLE] The Command statement of the action to which you are trying to add an action callback is different from the Command statement that started the application.

Implementation Specifics

OVwAddActionCallback supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddActionCallback and OVwRemoveActionCallback, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwAddAlertCallback(3)

Purpose

Registers handlers of NetView for AIX alerts

Related Functions

OVwRemoveAlertCallback

Syntax

```
#include <OV/ovw.h>

void (*OVwAlertCallbackProc) (void *userData,
       unsigned long alertClass, time_t alertTime,
       char *alertApp, char *alertMsg);

int OVwAddAlertCallback(unsigned long classMask,
       OVwAlertCallbackProc callback, void *userData);

int OVwRemoveAlertCallback(unsigned long classMask);
```

Note: These routines should be used only in an application solely designed for dispatching NetView for AIX alert messages that other applications have generated through OVwAlertMsg.

Description

OVwAddAlertCallback adds an application callback procedure for handling classes of NetView for AIX alert messages. The procedure is started when an application calls OVwAlertMsg with an alertClass that corresponds with the registered classMask.

OVwRemoveAlertCallback removes previously registered callbacks for the specified classMask.

Note: Use OVwAddAlertCallback and OVwRemoveAlertCallback only if your application's sole purpose is to dispatch NetView for AIX alert messages.

An application that has registered alert callbacks cannot use OVwAlertMsg.

Parameters

alertApp	Specifies a pointer to the name of the application that called OVwAlertMsg.
alertClass	Specifies the class of the alert message issued by alertApp.
alertMsg	Specifies a pointer to the text of the alert message issued by alertApp.
alertTime	Specifies the time slot when alertApp issued the alert message.
callback	Specifies a pointer to a procedure that starts in response to a NetView for AIX alert.
classMask	Specifies a logical OR of the classes of alerts for which the callback should be invoked. If classMask is 0 (zero), the callback is invoked for all alert classes.
userData	Specifies a pointer to application-specific data registered for the callback procedure.

Return Values

If successful, OVwAddAlertCallback and OVwRemoveAlertCallback return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddAlertCallback and OVwRemoveAlertCallback set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]
: The connection to the NetView for AIX program was lost.

[OVw_OUT_OF_MEMORY]
: A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
: The EUI API has not been initialized with OVwInit.

Examples

The following example is an application that displays NetView for AIX alert messages:

```
#include <OV/ovw.h>
#include <stdio.h>

void
alertProc (void *userData,
           unsigned long alertClass,
           time_t alertTime,
           char *alertApp,
           char *alertMsg)
{
    printf("%s: ", alertApp);
    switch (alertClass) {
      case ovwAlertInfo:
         printf("INFORMATION: ");
         break;
      case ovwAlertWarning:
         printf("WARNING: ");
         break;
      case ovwAlertError:
         printf("ERROR: ");
         break;
      case ovwAlertDisaster:
         printf("DISASTER: ");
         break;
      default:
         break;
    }
    printf("%s\n", alertMsg);
}
main(int argc, char **argv)
{
    if (OVwInit() < 0) {
        fprintf(stderr, "%s\n", OVwErrorMsg(OVwError ()));
        exit(1);
      }


      OVwAddCallback(ovwEndSession, NULL, (OVwCallbackProc)exit, NULL);
      OVwAddAlertCallback(0, alertProc, NULL);

      OVwMainLoop();

}
```

Implementation Specifics

OVwAddAlertCallback supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddAlertCallback or OVwRemoveAlertCallback, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAlertMsg(3)” on page 558.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwAddCallback(3)

Purpose

Registers procedures to process NetView for AIX events

Related Functions

OVwRemoveCallback

Syntax

```
#include <OV/ovw.h>

void (*OVwCallbackProc) (char *userData, OVwEventType event, ...);

int OVwAddCallback(OVwEventType event, OVwFieldBindList *capabilitySet,
       OVwCallbackProc proc, void *userData);

int OVwRemoveCallback(OVwEventType event, OVwFieldBindList *capabilitySet);
```

Description

OVwAddCallback adds an application callback procedure for handling the NetView for AIX events of an object with the specified capabilities.

OVwRemoveCallback removes a previously added callback procedure for a specified NetView for AIX event and set of capabilities.

Calls to OVwAddCallback for differing sets of capabilities are cumulative; multiple procedures can be registered for each NetView for AIX event with each event processing objects with different capabilities. Only one callback can be registered for a particular capabilitySet.

Parameters

event
Specifies the NetView for AIX event that the registered procedure should process.

capabilitySet
Specifies a pointer to a list of capability-field bindings that classify the objects for which this callback should be invoked. If a NULL value is supplied, the callback is enabled for all classes of objects that do not already have specific callbacks registered for them. If an event is not related to object manipulation, such as ovwEndSession or ovwMapOpen, this parameter is ignored.

callbackProc
Specifies a pointer to the procedure that should be invoked, for the specified event, for the specified objectType. The parameters of the callback procedure vary, based on the NetView for AIX event.

userData
Specifies a pointer to application-specific data to be passed to the callback procedure.

Return Values

If successful, OVwAddCallback and OVwRemoveCallback return code (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddCallback and OVwRemoveCallback set the error code-value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwAddCallback supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddCallback or OVwRemoveCallback, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwAddHelpCallback(3)

Purpose

Registers a handler for application help requests

Related Functions

OVwRemoveHelpCallback

Syntax

```
#include <OV/ovw.h>

void (*OVwHelpCallbackProc) (void *userData, unsigned long helpType,
       time_t requestTime, char *appName, char *appHelpDir,
       char *helpRequest);

int OVwAddHelpCallback (unsigned long typeMask,
       OVwHelpCallbackProc callback, void *userData);

int OVwRemoveHelpCallback(unsigned long typeMask);
```

Note: These routines should be used only in an application solely designed to respond to help requests that other applications have issued from OVwShowHelp to the NetView for AIX help system.

Description

OVwAddHelpCallback adds an application callback procedure for handling NetView for AIX help requests. The procedure is started when some application calls OVwShowHelp with a helpType that corresponds with the registered typeMask.

OVwRemoveHelpCallback removes previously registered callbacks for the specified typeMask.

Note: Use OVwAddHelpCallback and OVwRemoveHelpCallback only if your application's sole purpose is to implement the NetView for AIX help system.

An application that has registered help callbacks cannot use OVwShowHelp.

Parameters

appHelpDir
Specifies a pointer to the directory that the application has registered, through its application registration file, as its help directory.

appName
Specifies a pointer to the name of the application that called OVwShowHelp.

callback
Specifies a pointer to a procedure to call in response to an application help request.

helpRequest
Specifies a pointer to a string specifying the request issued by the application with OVwShowHelp.

helpType
Specifies the type of request issued by the application with OVwShowHelp.

requestTime
Specifies the time when the application issued the help request.

typeMask
Specifies a logical OR of the types of help requests for which the callback should be added. If typeMask is zero, the callback is added for all help types.

userData
Specifies a pointer to application-specific data to be passed to the callback procedure.

Return Values

If successful, OVwAddHelpCallback and OVwRemoveHelpCallback return code (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddHelpCallback and OVwRemoveHelpCallback set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwAddHelpCallback supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddHelpCallback or OVwRemoveHelpCallback, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwShowHelp(3)” on page 825.
- See “OVwApiIntro(5)” on page 560.

OVwAddInput(3)

Purpose

Adds an event source

Related Functions

OVwRemoveInput

Syntax

```
#include <OV/ovw.h>

void (*OVwInputCallbackProc) (int file_descriptor, void *userData);

OVwInputId OVwAddInput(int file_descriptor, int conditionMask,
       OVwInputCallbackProc proc, void *userData);

int OVwRemoveInput(OVwInputId id);
```

Description

OVwAddInput adds an application file descriptor to the NetView for AIX event processing mechanism as another source of events. When the specified condition occurs on the fileDescriptor, as detected by select(2), proc is called and is passed the parameter userData.

OVwRemoveInput removes an input event, created by a previous call to OVwAddInput, from the NetView for AIX event processing mechanism.

Parameters

fileDescriptor
Specifies the source file descriptor to be added to the NetView for AIX event processing loop.

conditionMask
Specifies a mask built from the following flags defined in the OV/ovw.h header file:

ovwReadMask There is input on the file descriptor.

proc
Specifies an application procedure to be started when the condition occurs on the fileDescriptor.

userData
Specifies a pointer to application-specific data registered when the input is added and passed when the callback is called.

id
Specifies the input ID returned from an earlier call to OVwAddInput.

Return Values

If successful, OVwAddInput and OVwRemoveInput return an OVwInputId. If unsuccessful, they return 0 (zero).

Error Codes

OVwAddInput and OVwRemoveInput set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwAddInput supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddInput or OVwRemoveInput, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwAddMenuItem(3)

Purpose

Adds a menu item to a menu

Related Functions

OVwRemoveMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddMenuItem(char *menuId, char **menuItemId);

int OVwRemoveMenuItem(char *menuId, char **menuItemId);
```

Description

OVwAddMenuItem associates a menu item with a registered menu or the graphical interface menu bar in the current registration context. See OVwSetRegContext in the man page for “OVwGetRegContext(3)” on page 729 for information about changing the registration context.

OVwRemoveMenuItem removes a menu item that is associated with a registered menu in the current registration context.

Before calling either of these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu structure will become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to dynamically create menu registration. If your application menu structure is static, use the application registration files to create the application menu structure.

Note: When you add a menu item to the menu bar, the menu attached to that menu item (through OVwAddMenuItemFunction) should have an ID that matches the menu item label. If it does not have a matching ID, inconsistent results can occur because the menu ID might not exist in subsequent NetView for AIX sessions.

Parameters

menuId
Specifies a pointer to a name for a menu as declared in the application registration file. If menuId is NULL, it refers to the graphical interface menu bar.

menuItemId
Specifies a pointer to a pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindMenuItem. The value of menuItemId can be modified after a successful call.

Return Values

If successful, OVwAddMenuItem and OVwRemoveMenuItem return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddMenuItem and OVwRemoveMenuItem set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENU_NOT_FOUND]	The argument menuId does not specify a valid menu.
[OVw_MENUITEM_NOT_FOUND]	The argument menuItemId does not specify a valid menu item.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	Either you have not called OVwLockRegUpdates prior to calling this function or you are adding a menu item to the menu bar that does not contain menu functions.

Implementation Specifics

OVwAddMenuItem supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddMenuItem or OVwRemoveMenuItem, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwAddMenuItemFunction(3)

Purpose

Adds a menu item function to a menu item

Related Functions

OVwRemoveMenuItemFunction

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddMenuItemFunction(char *menuItemId, int function,
       char *fnArg);

int OVwRemoveMenuItemFunction(char *menuItemId, int function,
       char *fnArg);
```

Description

OVwAddMenuItemFunction binds the specified function and argument to the specified menu item in the current registration context. See OVwSetRegContext in the man page for “OVwGetRegContext(3)” on page 729 for information about changing the registration context.

OVwRemoveMenuItemFunction removes the specified function and argument from the specified menu item in the current registration context.

Before calling either of these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu structure will become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to dynamically create menu registration. If your application menu structure is static, use the application registration files to create the application menu structure.

Note: When adding a menu function to a menu item that is, or will be, attached to the graphical interface menu bar (see “OVwAddMenuItem(3)” on page 545), the menu should have an ID that matches the menu item label. If it does not have a matching ID, inconsistent results can occur because the menu ID might not exist in subsequent NetView for AIX sessions.

Parameters

menuItemId
Specifies a pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindMenuItem.

fnArg
Specifies a pointer to the function argument whose meaning is determined by the function parameter.

function

Specifies the type of function you are binding to the menu item. These function types are defined in the OV/ovw.h header file as follows:

ovwMenu	The function argument fnArg is a menu identifier. For example, if fnArg is IP Commands, this would be equivalent to specifying f.menu IP Commands for the menu item in the application registration file.
ovwInternal	The function argument fnArg is an internal function name. For example, if fnArg is exit, this would be equivalent to specifying f.exit for the menu item in the application registration file.
ovwAction	The function argument fnArg is an action identifier. For example, if fnArg is Get, this would be equivalent to specifying f.action Get for the menu item in the application registration file.
ovwShell	The function argument fnArg is a shell command. For example, if fnArg is xterm -e /etc/ping ${OVwSelection1}, this would be equivalent to specifying ! xterm -e /etc/ping ${OVwSelection1} for the menu item in the application registration file.

Return Values

If successful, OVwAddMenuItemFunction and OVwRemoveMenuItemFunction return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwAddMenuItemFunction and OVwRemoveMenuItemFunction set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_FOUND]	The argument fnArg does not specify a valid action.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENUITEM_NOT_FOUND]	The argument menuItemId does not specify a valid menu item.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	Either you have not called OVwLockRegUpdates prior to calling this function or MenuItemId references a menu item that contains an incompatible function.
[OVw_MENU_NOT_FOUND]	The argument fnArg does not specify a valid menu.
[OVw_NAME_NOT_FOUND]	The argument fnArg does not specify a valid internal function name.

Implementation Specifics

OVwAddMenuItemFunction supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddMenuItemFunction or OVwRemoveMenuItemFunction, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddMenuItem(3)” on page 545.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwCreateMenu(3)” on page 611.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwAddObjMenuItem(3)

Purpose

Adds a menu item to an object menu

Related Functions

OVwRemoveObjMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddObjMenuItem(char *objMenuId, char **objMenuItemId);

int OVwRemoveObjMenuItem(char *objMenuId, char **objMenuItemId);
```

Description

OVwAddObjMenuItem ties a menu item to an object menu's registered menu or to the graphical interface object menu in the current registration context. See OVwSetRegContext in the man page for “OVwGetRegContext(3)” on page 729 for information about changing the registration context.

OVwRemoveObjMenuItem removes a menu item that is tied to an object menu's registered menu in the current registration context.

Before calling either of these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu structure will become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to dynamically create menu registration for the object menu. If your application menu structure is static, use the application registration files to create the application menu structure.

Note: When you add a menu item to the object menu, the menu attached to that menu item (through OVwAddMenuItemFunction) should have an ID that matches the menu item label. If it does not have a matching ID, inconsistent results can occur because the menu ID might not exist in subsequent NetView for AIX sessions.

Parameters

menuId
Specifies a pointer to a name for a menu as declared in the application registration file. If menuId is NULL, it refers to the graphical interface object menu.

menuItemId
Specifies a pointer to a pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindObjMenuItem. The value of menuItemId can be modified after a successful call.

Return Values

If successful, OVwAddObjMenuItem and OVwRemoveObjMenuItem return a value 0f 0 (zero). If unsuccessful, a value of –1 (negative one) is returned.

Error Codes

OVwAddObjMenuItem and OVwRemoveObjMenuItem set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENU_NOT_FOUND]	The argument *menuId* does not specify a valid menu.
[OVw_MENUITEM_NOT_FOUND]	The argument *menuItemId* does not specify a valid menu item.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	Either OVwLockRegUpdates has not been called prior to calling this function, or the menu item you are adding to the menu bar does not contain menu functions.

Implementation Specifics

OVwAddObjMenuItem supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddObjMenuItem or OVwRemoveObjMenuItem, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwAddObjMenuItemFunction(3)

Purpose

Adds a menu item function to an object menu's menu item

Related Functions

OVwRemoveObjMenuItemFunction

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddObjMenuItemFunction(char *objMenuItemId, int function,
       char *fnArg);

int OVwRemoveObjMenuItemFunction(char *objMenuItemId, int function,
       char *fnArg);
```

Description

The OVwAddObjMenuItemFunction routine binds the specified function and argument to the specified object menu's menu item in the current registration context. See OVwSetRegContext in the man page for “OVwGetRegContext(3)” on page 729 for information about changing the registration context.

OVwRemoveObjMenuItemFunction removes the specified function and argument from the specified object menu's menu item in the current registration context.

Before calling either of these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu structure become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to dynamically create menu registration for an object menu. If your application menu structure is static, use the application registration files to create the application menu structure.

Note: When adding a menu function to a menu item that is, or will be, attached to the graphical interface object menu (see “OVwAddObjMenuItem(3)” on page 550), the menu should have an ID that matches the menu item label. If it does not have a matching ID, inconsistent results can occur because the menu ID might not exist in subsequent NetView for AIX sessions.

Parameters

objMenuItemId
Specifies a pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindObjMenuItem.

fnArg
Specifies a pointer to the function argument whose meaning is determined by the function parameter.

function
Specifies the type of function you are binding to the menu item. These function types are defined in the OV/ovw.h header file as follows:

ovwMenu — The function argument *fnArg* is a menu identifier. For example, if *fnArg* is IP commands, this is equivalent to specifying `f.menu IP commands` for the menu item in the application registration file.

ovwInternal — The function argument *fnArg* is an internal function name. For example, if *fnArg* is exit, this is equivalent to specifying `f.exit` for the menu item in the application registration file.

ovwAction — The function argument *fnArg* is an action identifier. For example, if *fnArg* is Get, this is equivalent to specifying `f.action Get` for the menu item in the application registration file.

ovwShell — The function argument *fnArg* is a shell command. For example, if *fnArg* is xterm -e /etc/ping ${OVwSelection1}, this is equivalent to specifying `! xterm -e /etc/ping ${OVwSelection1}` for the menu item in the application registration file.

Return Values

If successful, OVwAddObjMenuItemFunction and OVwRemoveObjMenuItemFunction return a value of 0 (zero). If unsuccessful, a value of –1 (negative one) is returned.

Error Codes

OVwAddObjMenuItemFunction and OVwObjRemoveMenuItemFunction set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_FOUND] — The argument *fnArg* does not specify a valid action.

[OVw_CONNECTION_LOST] — The connection to the NetView for AIX program was lost.

[OVw_MENUITEM_NOT_FOUND] — The argument *menuItemId* does not specify a valid menu item.

[OVw_OUT_OF_MEMORY] — A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] — The EUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED] — Either OVwLockRegUpdates was not called prior to calling this function, or *MenuItemId* references a menu item that contains an incompatible function.

[OVw_MENU_NOT_FOUND] — The argument *fnArg* does not specify a valid menu.

[OVw_NAME_NOT_FOUND] — The argument *fnArg* does not specify a valid internal function name.

Implementation Specifics

OVwAddObjMenuItemFunction supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddObjMenuItemFunction or OVwRemoveObjMenuItemFunction, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddMenuItem(3)” on page 545.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwCreateMenu(3)” on page 611.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwAddToolPalItem(3)

Purpose

Adds a new item to the Tools window

Related Functions

OVwRemoveToolPalItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddToolPalItem(OVwToolPalRegInfo *itemInfo);

int OVwRemoveToolPalItem(char *toolItemId);
```

Description

OVwAddToolPalItem adds a specified item to the Tools window.

OVwRemoveToolPalItem removes the specified item from the Tool Palette.

Before calling either of these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the Tool Palette structure will become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to dynamically create Tools window registration. If your application Tools Window structure is static, use the application registration files to create the application Tools window structure.

Parameters

toolItemId
Specifies a pointer to the name of the Tools window item as defined in the application registration file for the current registration context or by the field label of the OVwToolPalRegInfo structure in a previous call to OVwAddToolPalItem.

itemInfo
Specifies a pointer to an OVwToolPalRegInfo structure. The OVwToolPalRegInfo structure contains the elements of the Tools window item registration information. In <OV/ovw_reg.h>, it is defined as shown in the following example:

```
typedef struct {
    int         precedence;
    char        *label;
    int         iconType;
    char        *iconColor;
    char        *iconFile;
    char        *dragBitmap;
    char        *labelColor;
    int         selMechanism;
    char        *actionId;
} OVwToolPalRegInfo;
```

The members of this structure are:

precedence
The precedence value for the tool item. Precedence values range from a low value ovwMinToolItemPrecedence (defined as 0 in <OV/ovw_reg.h>) to a high value of ovwMaxToolItemPrecedence (defined as 100 in <OV/ovw_reg.h>). If no specific precedence is needed for the menu item, set this field to ovwDefaultToolItemPrecedence (defined as 50 in <OV/ovw_reg.h>).

label A pointer to the tool item label string.

iconType The type of image used for of the tool item in the Tools window. The value is one of the following constants defined in <OV/ovw_reg.h>):

```
ovwIcSolid          "solid"
ovwIcGif            "gif"
ovwIcBitmap         "bitmap"
```

iconColor
A pointer to the color name to be applied to the icon in the Tools window.

iconFile The name of the file that contains the icon image of the tool item to the displayed in the Tools window.

dragBitmap
The name of the file that contains the icon bitmap of the tool item that is displayed in drag operations.

labelColor
A pointer to the color name to be applied to the label string.

selMechanism
Specifies the options for the selection mechanism to be applied to the tool item. The value is a mask that is the logical OR of the following constants from <OV/ovw_reg.h>:

ovwSingleClick
Specifies that the user can select the item through a single-click operation.
ovwDoubleClick
Specifies that the user can select the item through a double-click operation.
ovwDragDrop
Specifies that the user can select the item through a drag and drop operation.

actionId Specifies a pointer to the name of the action as defined in the application registration file for the current registration context.

Return Values

If successful, OVwAddToolPalItem and OVwRemoveToolPalItem return a value of 0 (zero). If unsuccessful, a value of –1 (negative one) is returned.

Error Codes

OVwAddToolPalItem and OVwRemoveToolPalItem set the error-code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_FOUND] The specified actionId is not registered in the current registration context.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED] You have not called OVwLockRegUpdates prior to calling this function.

[OVw_TOOLITEM_EXISTS] The specified tool item is already registered in the current registration context.

[OVw_TOOLITEM_INFO_NULL] The argument itemInfo is a null pointer.

[OVw_TOOLITEM_NOT_FOUND] The argument toolItemId is not registered in the current registration context.

[OVw_TOOLITEM_PRECEDENCE_ERROR] The specified precedence is not within the valid range of precedence values.

[OVw_TOOLITEM_MECHANISM_ERROR] The specified selection mechanism is not within the valid range of selection mechanism values.

Implementation Specifics

OVwAddToolPalItem supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAddToolPalItem or OVwRemoveToolPalItem, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwAlertMsg(3)

Purpose

Issues an NetView for AIX alert message

Syntax

```
#include <OV/ovw.h>

int OVwAlertMsg(unsigned long alertClass, char *message);
```

Description

OVwAlertMsg provides a way for applications to present messages to the user. It sends a message and severity classification to an application responsible for receiving NetView for AIX alerts and presenting them to the user.

The number of events that OVwAlertMsg can send within a short period of time is limited. If it sends the maximum number of events, it will not be able to emit further events. To prevent OVwAlertMsg from reaching the limit, allow a short amount of time to elapse between calls to OVwAlertMsg.

When applications use OVwAlertMsg to generate NetView for AIX alert messages, OVwAddAlertCallback and OVwRemoveAlertCallback should be used in an application solely designed for dispatching these alert messages.

Parameters

alertClass
Specifies a message classification. The following permitted values of alertClass are specified in the header file OV/OVw.h:

ovwAlertDisaster	Refers to a failure that compromises the application and makes further normal operation impossible.
ovwAlertError	Refers to a failure affecting only a particular operation.
ovwAlertWarning	Refers to an event that might be a failure; a case where the application will attempt to complete the operation anyway. A warning message should generally be issued only when it suggests some action the administrator can take to reduce the likelihood of a future problem.
ovwAlertInfo	Reports significant but normal events. The information provided should be just sufficient for the administrator to reconstruct a history of operations that have taken place.

message
Specifies a pointer to the text of the alert message.

Return Values

If successful, OVwAlertMsg returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwAlertMsg sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	There is no ovw application running that is dispatching alert messages.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwAlertMsg supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwAlertMsg, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwAddAlertCallback(3)” on page 535.

OVwApiIntro(5)

Purpose

Provides an overview of the EUI API

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

cc < or CC or ld > ... -lovw -lov -lntl
```

Description

The EUI API contains a registration facility and a set of library routines that enable applications to integrate with the NetView for AIX program, the NetView for AIX network, and the systems management user interface. The ovw command provides the graphical interface for integrating network and systems management applications through a common user interface and shared graphical map of the management environment.

Different types of applications will use different parts of the EUI API. Loosely integrated applications that are merely started through a menu item will only need to use the application registration facility described in the following information under Registration Files on page 562. Most applications will use error-handling, process-management, event-processing, and general routines. Specialized map applications that dynamically update the map will additionally use object, symbol-type, and map routines, as well as map editing.

Concepts: The following concepts are important for understanding the EUI API. See the *NetView for AIX Programmer's Guide* for a more detailed introduction to these concepts.

Action
: Specifies a registered operation performed by an application. An action can be associated with a menu item or an executable symbol. When an action is invoked by a user, the action and a list of selected objects on which to perform the action are passed to the application.

Application
: Specifies a registered program that is started by the ovw command. Some applications perform actions associated with menu items. Other applications dynamically update the graphical map to reflect the current state of the management environment.

Field
: Specifies a global object attribute. Field values for objects are stored in the OVW object database.

  A capability field, such as isRouter and isIP, is a special kind of field that indicates an important attribute for classifying objects. A field can be designated as a capability field upon creation. An object, which can have values for multiple capability fields, can serve multiple functions. Capability fields are used for the following functions:

  - Menu greying. For example, a selection rule can be defined so that a menu item is enabled only when certain kinds of objects are selected.
  - Determining what field information is presented for an object for editing operations, such as add, connect, describe
  - Filtering EUI API events.

A name field, such as Selection Name or IP Host name, is a special kind of field that uniquely identifies an object. A field can be designated as a name field upon creation. A name field has a value that is unique for each object in the database.

A selection name is a name field that is the only required textual name for an object. This is the principal name by which an object is known through the user interface. A user can set the selection name for an object to any one of its multiple names.

Map
: Specifies a named collection of objects and submaps. The containment relationship of the objects on the map is displayed through a hierarchy of submaps. Different maps can represent different administrative or management domains or different presentations of the same management environment.

Object
: Specifies that a graphical interface object is a construction that represents a particular entity or resource in the management environment, such as a network, a host, or a process. An object is a semantic element that exists across maps. The same object can be represented by multiple symbols on multiple maps. Objects and their field values (attributes) are stored in the OVW object database. Objects are used to present information about management resources through the user interface. An object has global attributes (object ID and field values) that exist across maps and certain characteristics (child submap, status) that are specific to a particular map on which the object can exist. For the most part, an object exists on a map when it has an associated symbol on the map.

  A parent object is an object that has an associated child submap on a particular map. It is also called a compound object.

Symbol
: Specifies a graphical representation of an object. A symbol represents a particular object as it appears on a submap of a particular map. An object can be represented by multiple symbols. Multiple symbols for the same object can exist on the same submap, on different submaps of the same map, and across maps. Symbols are presentation elements for displaying objects and have characteristics including variety, behavior, label, status, status source, and symbol type. The following list describes different types of symbols:

  Executable symbol
  : Invokes an application action when double -clicking on the symbol

  Explodable symbol
  : Opens into the child submap of the parent object represented by the symbol when double-clicking on the symbol.

  Icon symbol
  : Is depicted with a bitmap graphic

  Connection symbol
  : Is depicted as a line connecting two other symbols

  Component symbol
  : Is a symbol on the child submap of a compound object

Symbol type
: Specifies a characteristic of a symbol that specifies its visual appearance. Symbol type values are registered through a symbol type registration file. A symbol type has two components: symbol class symbol subclass. A symbol type value is described with a string in the form <class>:<subclass> (for example, Computer:Workstation). For icon symbols, the symbol class determines the outside shape for depicting the symbol and the subclass determines the inside bitmap. Symbol types also have default capability field values associated with them that are used to initialize capability field values on the object of the symbol with the given symbol type.

Submap
: Specifies a collection of related symbols displayed together in a single window. A submap shows a particular view of the management environment. The following list describes different types of submaps:

Child submap
: Has an associated parent object on a particular map. A child submap provides a detailed view of the contents of the parent object.

Orphan submap
: Does not have an associated parent object.

Shared submap
: Can be updated by any application.

Exclusive submap
: Can be updated only by the application that created the submap.

Metaconnection submap
: Is a special submap automatically generated by the graphical interface to show multiple connections between two symbols.

Registration Files: Registration files are used to define static configuration information. This information is read when the ovw command is run with an installation option or when the graphical interface is started. Registration is accomplished by creating a file with the appropriate format and placing it in a special directory. There are three kinds of registration files:

Application
: Application registration files are found in the /usr/OV/registration/$LANG directory. An application entry specifies the actions that an application can perform, a command line to invoke the application, flags indicating how the application process should be started and managed, menu and menu item definitions, descriptions of automatically-generated editing dialog boxes for querying and displaying object fields, and other application information. The application registration files are read when the graphical interface is started.

Field
: Field registration files are found in the /usr/OV/lib/fields/$LANG directory. A field entry specifies the field name, the field data type (boolean, string, enumerated type, integer), and flags describing how the field is to be used (for example, list or capability). When a change is made to the field registration directory, issue the following command to create the registered fields in the OVW object database:

```
ovw -fields
```

Symbol Type
: Symbol type registration files are found in the /usr/OV/lib/symbols/$LANG directory. The following list describes symbol type entries in registration files:

  Symbol class entry
  : Specifies the class name and, for an icon or connection, the variety of the symbol class. For icons, a symbol class entry also provides a graphical specification for the outside shape.

  Symbol type entry
  : Specifies the symbol subclass name, the symbol class to which the symbol type belongs, the bitmaps (for icons) or line style (for connections) to use display, a set of default capability field values for initializing objects represented by symbols with the symbol type, and other symbol type information.

  The symbol type registration files are read whenever the graphical interface is started. You can speed up the startup time for the graphical interface by running the following command when new symbol types are registered by issuing the following command:

  ```
  ovw -config
  ```

  This command compiles all the bitmap files for icon symbol types.

Running the following command verifies the syntax of all the registration files:

```
ovw -verify
```

See “OVwRegIntro(5)” on page 769 for the details of registering applications, fields, and symbol types.

Error Handling: The return code of a function indicates whether the call succeeded or an error was encountered, as shown in Table 18 on page 563.

Table 18. Return Codes

Type of Return Code	Code for Success	Code for Failure
Integer value	0 (zero)	–1
Pointer	Valid pointer	NULL
ID values	Valid ID	NULL ID

Note: The macro OVwIsIdNull should be used to check for a NULL ID value.

The error codes for the EUI API are listed in the <OV/ovw_errs.h> header file.

The following routines are error handling routines:

OVwError Returns the last EUI API error

OVwErrorMsg Returns the text for an OVwError error code

Process Management: An application must call either OVwInit or OVwDbInit prior to using the EUI API. OVwInit establishes communication with the NetView for AIX program from which the application was started. OVwDbInit establishes communication with the ovwdb daemon process for access to the OVW object database. OVwInit automatically calls OVwDbInit. A program should call OVwDbInit directly only if it needs to use the NetView for AIX object database routines, those routines beginning with OVwDb, but does not need to use the rest of the EUI API or to be started from the NetView for AIX program.

The following routines are process management routines:

OVwInit Initializes a connection to the NetView for AIX program

OVwDbInit Initializes a connection between the ovwdb daemon process

OVwDone Terminates the connection to the NetView for AIX program

Event Processing: An application can register to receive various asynchronous events from the NetView for AIX program. This is done by registering a callback function to be called when the event occurs.

The function OVwAddCallback can be used to register for various EUI API events, such as when the graphical interface exits or when a new map is opened. OVwAddCallback is called with an event type, a function conforming to the callback type for the indicated event type, and an optional filter based on capability field values. See “OVwMapOpenCB(3)” on page 761 for an example of registering for an EUI API event. The EUI API events and the corresponding callbacks are defined in the header file <OV/ovw.h> and described in “OVwEventIntro(5)” on page 691.

Applications can also register to receive notification when an action registered by the application through an application registration file is invoked by an user. An action is invoked when the user selects a menu item or double-clicks on an executable symbol. The routine OVwAddActionCallback is used to register callback functions to be called when an action is requested.

The following routines are used to register callbacks for EUI API events and actions:

OVwAddCallback Add a callback for an EUI API event

OVwRemoveCallback Remove a callback for an EUI API event

OVwAddActionCallback	Add a callback for an action request
OVwRemoveActionCallback	Remove a callback for an action request

In order for events to be received from the NetView for AIX program, an application must register for the events and enter an event processing loop after finishing initialization. An application can use one of the following approaches:

- An application that needs to process X events can call OVwXtMainLoop or OVwXtAppMainLoop. These calls will enable an application to get both NetView for AIX events and X events. Use XtAddInput to include the NetView for AIX input source and then call XtMainLoop or XtAppMainLoop. Use OVwXtAddInput or OVwXtAppAddInput to include additional input sources in the event processing loop.
- An application that does not need to process X events can call OVwMainLoop to receive NetView for AIX events. Use OVwAddInput to include additional input sources in the event processing loop.
- An application that needs to perform select(2) processing can use OVwFileDescriptor and OVwProcessEvent to get NetView for AIX events. Most applications do not need to use these low-level routines.

The following routines are event processing routines:

OVwMainLoop	Continuously processes NetView for AIX events
OVwAddInput	Adds a callback for input on a file descriptor for use with OVwMainLoop
OVwRemoveInput	Removes a callback for input on a file descriptor for use with OVwMainLoop
OVwPeekOVwEvent	Determines whether a particular NetView for AIX event is pending
OVwPeekInputEvent	Determines whether a particular input event is pending
OVwXtMainLoop	Continuously processes X and NetView for AIX events
OVwXtAppMainLoop	Continuously processes X and NetView for AIX events
OVwXtAddInput	Adds a callback for input on a file descriptor for use with OVwXtMainLoop or OVwXtAppMainLoop
OVwXtAppAddInput	Removes a callback for input on a file descriptor for use with OVwXtMainLoop or OVwXtAppMainLoop
OVwPending	Determines whether any NetView for AIX event is pending
OVwProcessEvent	Processes the next NetView for AIX event
OVwFileDescriptor	Gets the NetView for AIX input file descriptor

General Routines:

The following routines are general routines:

OVwGetAppName	Gets the name of the calling application
OVwGetSelections	Gets the current object selection list
OVwHighlightObject	Highlights an object on the open map
OVwHighlightObjects	Highlights objects on the open map
OVwShowHelp	Displays an application help screen
OVwAlertMsg	Generates an alert message

Object Routines: The routines that access the OVW object database all begin with the prefix OVwDb and are defined in the <OV/ovw_obj.h> header file. These routines provide access to global object information; they are not map-specific.

When testing or comparing object IDs and field IDs, use the macros OVwIsIdNull and OVwIsIdEqual.

A field can be created either through a field registration file or the OVwCreateField routine. The OVwCreateField routine is not needed for fields registered through a field registration file.

The following routines are field routines:

OVwDbCreateField	Creates a new field
OVwDbDeleteField	Deletes a field
OVwDbGetFieldInfo	Gets field information
OVwDbFreeFieldInfo	Frees field information
OVwDbListFields	Gets a list of fields
OVwDbFreeFieldList	Frees a list of fields
OVwDbFieldNameToFieldId	Converts field name to field ID
OVwDbFieldIdToFieldName	Converts field ID to field name
OVwDbGetEnumConstants	Gets name constants for an enumerated type
OVwDbFreeEnumConstants	Frees name constants for an enumerated type
OVwDbSetEnumConstants	Sets name constants for an enumerated type
OVwDbAppendEnumConstants	Appends name constants for an enumerated type
OVwDbGetEnumValue	Converts a name constant to an enumerated type value
OVwDbGetEnumName	Converts an enumerated type value to a name constant

An object can be created directly through one of the object creation routines. An object can also be created indirectly through the symbol creation routines, which do automatic object creation.

The following routines are object routines:

OVwDbCreateObject	Creates a graphical interface object
OVwDbCreateObjectBySelectionName	Creates a graphical interface object with a selection name
OVwDbCreateObjectByHostname	Creates a graphical interface object with an IP hostname
OVwDbDeleteObject	Deletes a graphical interface object

The following routines are object routines that access fields:

OVwDbGetFieldValue	Gets a particular field value for an object
OVwDbFreeFieldValue	Frees a field value
OVwDbGetFieldValues	Gets a list of all field values for an object
OVwDbFreeFieldBindList	Frees a field value list
OVwDbGetCapabilityFieldValues	Gets a list of capability field values for an object
OVwDbGetNameFieldValues	Gets a list of name field values for an object
OVwDbSetFieldValue	Sets the value of a field for an object

OVwDbSetSelectionName	Sets the selection name of an object
OVwDbSetHostname	Sets the IP hostname of an object
OVwDbUnsetFieldValue	Removes the value of a field for an object
OVwDbUnsetFieldValues	Removes the values for a list of fields for an object
OVwDbGetUniqObjectName	Gets a unique value for a name field
OVwDbNameToObjectId	Converts a name field value to object ID
OVwDbListObjectsByFieldValue	Locates all objects with a field value
OVwDbListObjectsByFieldValues	Locates all objects with field values
OVwDbFreeObjectIdList	Frees an object ID list
OVwDbGetFieldValuesByObjects	Gets the value of a field for each object
OVwDbFreeObjectFieldList	Frees an object field list
OVwDbSelectionNameToObjectId	Converts selection name to object ID
OVwDbObjectIdToSelectionName	Converts object ID to selection name
OVwDbHostnameToObjectId	Converts IP hostname to object ID
OVwDbObjectIdToHostname	Converts object ID to IP hostname

Symbol Type Routines: The following routines are symbol type routines:

OVwListSymbolTypes	Gets a list of all registered symbol types
OVwListSymbolTypeCaps	Gets the default capabilities for a symbol type

The symbol type routines enable you to programmatically determine which symbol type to use for displaying an object, based on the default capabilities that are associated with that symbol type.

Map Routines: Users create maps and control their opening and closing through the graphical interface. Through application configuration and use of the manage and unmanage operations, users control what information about the management environment will be displayed by map applications on a map. Users can also edit the map to supplement or modify information presented by applications.

A map contains submaps, which, in turn, contain symbols. An object appears on a map by being represented by a symbol on that map. However, in certain cases, an object can exist on a map without being represented by a symbol on the map.

When testing or comparing submap IDs and symbol IDs, use the macros OVwIsIdNull and OVwIsIdEqual.

There are a number of EUI API events that an application can receive to monitor changes in the map. See “OVwEventIntro(5)” on page 691 for a complete list.

The following map routines operate only on an open map:

OVwGetMapInfo	Gets map information
OVwCopyMapInfo	Copies map information
OVwFreeMapInfo	Frees map information
OVwBeginMapSync	Begins the map synchronization phase
OVwEndMapSync	Ends the map synchronization phase
OVwAckMapClose	Acknowledges a map close event

OVwGetAppConfigValues	Gets application configuration values for a map
OVwSetAppConfigValues	Sets application configuration values for a map

The following routines are submap routines:

OVwCreateSubmap	Creates a submap
OVwDeleteSubmap	Deletes a submap
OVwGetSubmapInfo	Gets submap information
OVwFreeSubmapInfo	Frees submap information
OVwListSubmaps	Gets a list of submaps
OVwFreeSubmapList	Frees a list of submaps
OVwDisplaySubmap	Displays a submap
OVwSetSubmapName	Sets the name of a submap
OVwSetBackgroundGraphic	Sets the background graphic for a submap
OVwClearBackgroundGraphic	Clears the background graphic for a submap

The following routines are the symbol routines:

OVwCreateSymbol	Creates a symbol
OVwCreateSymbols	Creates multiple symbols
OVwCreateSymbolByName	Creates a symbol with an object name
OVwCreateSymbolBySelectionName	Creates a symbol with a selection name
OVwCreateSymbolByHostname	Creates a symbol with an IP hostname
OVwCreateComponentSymbol	Creates a component symbol
OVwCreateComponentSymbolByName	Creates a component symbol with an object name
OVwCreateConnSymbol	Creates a connection symbol
OVwCreateConnSymbolByName	Creates a connection symbol with an object name
OVwDeleteSymbol	Deletes a symbol
OVwDeleteSymbols	Deletes multiple symbols
OVwGetSymbolInfo	Gets symbol information
OVwFreeSymbolInfo	Frees symbol information
OVwGetConnSymbol	Gets a connection symbol
OVwGetSymbolsByObject	Gets a list of symbols representing an object
OVwListSymbols	Gets a list of symbols on a submap
OVwFreeSymbolList	Frees a symbol list
OVwSetStatusOnSymbol	Sets the status of a symbol
OVwSetStatusOnSymbols	Sets the status of multiple symbols
OVwSetSymbolStatusSource	Sets the status source of a symbol
OVwSetSymbolLabel	Sets the label of a symbol
OVwSetSymbolApp	Expresses application interest in a symbol

OVwClearSymbolApp	Clears application interest in a symbol
OVwSetSymbolType	Changes the symbol type of a symbol
OVwSetSymbolPosition	Moves a symbol
OVwSetSymbolBehavior	Makes a symbol explodable or executable

The OVwObjectInfo structure, defined in the <OV/ovw.h> header file, describes an object as it exists on a particular map.

The following routines are map-specific object routines:

OVwGetObjectInfo	Gets map-specific object information
OVwFreeObjectInfo	Frees map-specific object information
OVwListObjectsOnMap	Gets a list of objects on the open map
OVwFreeObjectList	Frees an object list
OVwSetStatusOnObject	Sets the status of an object
OVwSetStatusOnObjects	Sets the status of multiple objects

Map Editing: Some user-editing operations, such as managing or unmanaging an object or moving a symbol, cause a notification event to be sent to applications that have registered for the event. See "OVwEventIntro(5)" on page 691 for a list of these events.

Other user-editing operations allow application validation of the operation to ensure its semantic correctness. This is accomplished by the following interaction:

1. The NetView for AIX program generates a query event, such as ovwQueryAddSymbol.
2. An application responds by calling a verification routine, such as OVwVerifyAdd.
3. The NetView for AIX program generates a final confirm event, such as ovwConfirmAddSymbol, to notify the application of the operation results. See "OVwVerifyAdd(3)" on page 833 for more details about this interaction.

See the *NetView for AIX Programmer's Guide* for more information on the query-verify-confirm sequence.

The following routines are map editing verification routines:

OVwVerifyAdd	Verifies user add of an object
OVwVerifyConnect	Verifies user connect of two symbols
OVwVerifyDescribeChange	Verifies user change of object description information
OVwVerifyDeleteSymbol	Verifies user delete of symbols
OVwVerifyAppConfigChange	Verifies user change of application configuration values

Header Files:

<OV/ovw.h> This header file is the main header file for the EUI API. It defines most of the EUI API structures and routines. If includes the following header files:

- <OV/ovw_types.h>
- <OV/ovw_obj.h>
- <OV/ovw_errs.h>
- <OV/ovw_string.h>

Because it includes other header files, the <OV/ovw.h> header file is the only header file that you need to include to use the EUI API, unless you are using programmatic application registration.

<OV/ovw_errs.h> This header file defines the error codes returned by OVwError.

<OV/ovw_fields.h> This header file defines string constants for predefined fields registered in the field registration file /usr/OV/lib/fields/c/ovw_fields.

<OV/ovw_obj.h> This header file defines structures and routines for accessing the OVW object database.

<OV/ovw_reg.h> This header file defines routines for performing programmatic application registration. Because file registration can be used to do application registration, few applications will need to use these routines.

<OV/ovw_string.h> This header file defines string constants useful for developers. It includes the <OV/ovw_fields.h> and <OV/sym_types.h> header files.

<OV/ovw_types.h> This header file defines some basic types for the EUI API, such as OVwBoolean and OVwStatusType. It also defines the macros OVwIsIdEqual and OVwIsIdNull for testing and comparing IDs.

<OV/sym_types.h> This header file defines string constants for predefined symbol types registered in certain symbol type registration files in the /usr/OV/lib/symbols/c directory.

Files

<OV/ovw.h>	Header file for the EUI API
<OV/ovw_errs.h>	Header file for the EUI API errors
<OV/ovw_fields.h>	Header file for predefined fields
<OV/ovw_obj.h>	Header file for object database routines
<OV/ovw_reg.h>	Header file for application registration
<OV/ovw_string.h>	Header file defining string constants
<OV/ovw_types.h>	Header file for certain EUI API types
<OV/sym_types.h>	Header file for predefined symbol types
/usr/OV/bitmaps/*	Symbol type bitmap directories
/usr/OV/databases/mapdb/*	Map database directories
/usr/OV/databases/ovwdb/*	Object database directories
/usr/OV/fields/*	Field registration directories

/usr/OV/help/*	Online help directories
/usr/OV/registration/*	Application registration directories
/usr/OV/symbols/*	Symbol type registration directories

Libraries

When compiling a program that uses the EUI API, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See “OVwAckMapClose(3)” on page 528.
- See “OVwAckUserSubmapCreate(3)” on page 530.
- See “OVwCreateAction(3)” on page 603.
- See “OVwAddActionCallback(3)” on page 532.
- See “OVwAddAlertCallback(3)” on page 535.
- See “OVwAddCallback(3)” on page 539.
- See “OVwAddHelpCallback(3)” on page 541.
- See “OVwAddInput(3)” on page 543.
- See “OVwAddMenuItem(3)” on page 545.
- See “OVwAddMenuItemFunction(3)” on page 547.
- See “OVwAddObjMenuItem(3)” on page 550.
- See “OVwAddToolPalItem(3)” on page 555.
- See “OVwAlertMsg(3)” on page 558.
- See “OVwCreateApp(3)” on page 607.
- See “OVwBeginMapSync(3)” on page 573.
- See “OVwConfirmCapabilityChangeCB(3)” on page 580.
- See “OVwConfirmCreateObjectsCB(3)” on page 582.
- See “OVwConfirmCreateSubmapsCB(3)” on page 584.
- See “OVwConfirmCreateSymbolsCB(3)” on page 586.
- See “OVwConfirmDeleteObjectsCB(3)” on page 588.
- See “OVwConfirmDeleteSubmapsCB(3)” on page 590.
- See “OVwConfirmHideSymbolsCB(3)” on page 594.
- See “OVwConfirmManageObjectsCB(3)” on page 596.
- See “OVwConfirmMoveSymbolCB(3)” on page 599.
- See “OVwConfirmObjectStatusCB(3)” on page 601.
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwDbAppendEnumConstants(3)” on page 633.
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbDeleteObject(3)” on page 641.
- See “OVwDbFieldNameToFieldId(3)” on page 643.
- See “OVwDbGetFieldInfo(3)” on page 648.
- See “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbGetFieldValues(3)” on page 654.
- See “OVwDbGetFieldValuesByObjects(3)” on page 656.
- See “OVwDbGetUniqObjectName(3)” on page 658.

- See “OVwDbHostnameToObjectId(3)” on page 660.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbListFields(3)” on page 664.
- See “OVwDbListObjectsByFieldValue(3)” on page 667.
- See “OVwDbNameToObjectId(3)” on page 670.
- See “OVwDbSelectionNameToObjectId(3)” on page 672.
- See “OVwDbSetEnumConstants(3)” on page 674.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbSetSelectionName(3)” on page 679.
- See “OVwDbUnsetFieldValue(3)” on page 681.
- See “OVwDisplaySubmap(3)” on page 683.
- See “OVwDone(3)” on page 685.
- See “OVwEndSessionCB(3)” on page 686.
- See “OVwError(3)” on page 688.
- See “OVwErrorMsg(3)” on page 689.
- See “OVwFileDescriptor(3)” on page 694.
- See “OVwFindMenuItem(3)” on page 696.
- See “OVwGetAppConfigValues(3)” on page 698.
- See “OVwGetAppName(3)” on page 701.
- See “OVwGetConnSymbol(3)” on page 702.
- See “OVwGetFirstAction(3)” on page 706.
- See “OVwGetFirstMenuItem(3)” on page 708.
- See “OVwGetFirstMenuItemFunction(3)” on page 710.
- See “OVwGetFirstRegContext(3)” on page 717.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwGetMenuItemPath(3)” on page 721.
- See OVwSetMenuPathSeparator in “OVwGetMenuPathSeparator(3)” on page 723.
- See “OVwGetObjectInfo(3)” on page 725.
- See “OVwGetRegContext(3)” on page 729.
- See “OVwGetSelections(3)” on page 731.
- See “OVwGetSubmapInfo(3)” on page 733.
- See “OVwGetSymbolInfo(3)” on page 735.
- See “OVwGetSymbolsByObject(3)” on page 737.
- See “OVwHighlightObject(3)” on page 739.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See “OVwListObjectsOnMap(3)” on page 745.
- See “OVwListSubmaps(3)” on page 747.
- See “OVwListSymbols(3)” on page 750.
- See “OVwListSymbolTypes(3)” on page 753.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwMainLoop(3)” on page 757.
- See “OVwMapCloseCB(3)” on page 759.
- See “OVwMapOpenCB(3)” on page 761.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwCreateMenu(3)” on page 611.
- See “OVwPeekOVwEvent(3)” on page 763.
- See “OVwPending(3)” on page 765.
- See “OVwProcessEvent(3)” on page 767.
- See “OVwRenameRegContext(3)” on page 795.
- See “OVwSelectListChangeCB(3)” on page 799.
- See “OVwSetBackgroundGraphic(3)” on page 801.
- See “OVwSetStatusOnObject(3)” on page 803.
- See “OVwSetSubmapName(3)” on page 806.

- See “OVwSetSymbolApp(3)” on page 808.
- See “OVwSetSymbolBehavior(3)” on page 810.
- See “OVwSetSymbolLabel(3)” on page 813.
- See “OVwSetSymbolPosition(3)” on page 815.
- See “OVwSetSymbolStatusSource(3)” on page 820.
- See “OVwSetSymbolType(3)” on page 822.
- See “OVwShowHelp(3)” on page 825.
- See “OVwSubmapCloseCB(3)” on page 827.
- See “OVwSubmapOpenCB(3)” on page 829.
- See “OVwUserSubmapCreateCB(3)” on page 831.
- See “OVwVerifyAdd(3)” on page 833.
- See “OVwVerifyAppConfigChange(3)” on page 839.
- See “OVwVerifyConnect(3)” on page 842.
- See “OVwVerifyDeleteSymbol(3)” on page 846.
- See “OVwVerifyDescribeChange(3)” on page 849.
- See “OVwXtAddInput(3)” on page 852.
- See “OVwXtMainLoop(3)” on page 855.
- See “OVwEventIntro(5)” on page 691.
- See “OVwRegIntro(5)” on page 769.
- See *NetView for AIX Application Interface Style Guide*.
- See *NetView for AIX Programmer's Guide*.
- See *NetView for AIX User's Guide for Beginners*.

OVwBeginMapSync(3)

Purpose

Begins map synchronization phase

Related Functions

OVwEndMapSync

Syntax

```
#include <OV/ovw.h>

int OVwBeginMapSync(OVwMapInfo *map);

int OVwEndMapSync(OVwMapInfo *map);
```

Description

OVwBeginMapSync marks the beginning of a synchronization phase that typically occurs when a map is opened. When a map is opened, a map application will normally update it to reflect status and topology changes that have occurred since the last time the map was closed. Calling OVwBeginMapSync will result in an indication on the status line below the graphical interface windows that map synchronization is in progress. During this period, map information might not yet be up-to-date.

OVwEndMapSync marks the end of the map synchronization phase. OVwEndMapSync should be called once for every call to OVwBeginMapSync to clear the indication that map synchronization is in progress.

OVwBeginMapSync and OVwEndMapSync can be used by multiple applications. The map synchronization indication is cleared when the last application calls OVwEndMapSync.

Note: If fewer calls are made to OVwEndMapSync than to OVwBeginMapSync, the map synchronization indication will not be cleared.

Parameters

map
Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

Return Values

If successful, OVwBeginMapSync and OVwEndMapSync return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwBeginMapSync and OVwEndMapSync set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to ovw was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.

[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following example is a callback routine for an ovwMapOpen event:

```
void
mapOpenProc(void *user_data, OVwEventType type, OVwMapInfo *map,
       OVwFieldBindList *config_params)
{
       OVwBeginMapSync(map);

      /* update status of objects and symbols on the map */

      if (map→permissions == ovwMapReadWrite) {
           /*
            * Update map for topology changes occurring
            * since map→last_closed_time.
            */
       }

       OVwEndMapSync(map);
}
```

Implementation Specifics

OVwBeginMapSync supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwBeginMapSync or OVwEndMapSync, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwMapOpenCB(3)” on page 761.
- See “OVwApiIntro(5)” on page 560.

OVwCheckAction(3)

Purpose

Enables applications to check the validity of the NetView for AIX application actions

Related Functions

OVwDoAction

Syntax

```
#include <OV/ovw.h>

OVwBoolean OVwCheckAction(char *appName, char *actionId,
                OVwObjectIdList *selections,
                OVwMapInfo *map, OVwSubmapId submap);

int OVwDoAction(char *appName, char *actionId,
                OVwObjectIdList *selections,
                OVwMapInfo *map, OVwSubmapId submap);
```

Description

OVwCheckAction enables an application to determine whether another application's action is applicable to the specified selection list.

OVwDoAction invokes an application action on the specified selection list.

These APIs permit applications to trigger NetView for AIX applications in the same manner in which they are triggered from the graphical interface menu bar or from executable symbols.

Parameters

appName
Specifies a pointer to the name of the NetView for AIX application which defines the specified action. If appName is NULL, it is assumed that the specified actionId is registered with the application making the call.

actionId
Specifies a pointer to the name of an action registered for the specified application.

selections
Specifies a pointer to a list of target object IDs for the application action.

map
Specifies a pointer to a MapInfo structure for the map on which the target objects are located. This information is supplied directly to the triggered application's action callback. A NULL map is permitted if it is not critical to the functions of the application's action. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submap
Specifies the ID of the submap where the target objects are located. This information is supplied directly to the triggered application's action callback. A NULL submap ID is permitted if it is not critical to the functions of the application's action.

Return Values

If the application action is valid for the specified selections, map, and submap, OVwCheckAction returns 0. If the application action is not valid or available, it returns -1.

If successful, OVwDoAction returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwCheckAction and OVwDoAction set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_APPLICABLE] This error code is returned if any of the following conditions are true.

- The objects specified by selections do not meet the selection list requirements of the action specified by actionId. See the registration file definition of the action to find its selection list requirements.
- There is no command specifying the action.
- The command does not match the application's command.

[OVw_ACTION_NOT_FOUND] The action specified by actionId is not registered by the application appName.

[OVw_APP_NOT_FOUND] The application appName is not a registered application.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_SUBMAP_NOT_FOUND] The submap specified by submap does not exist on the open map.

Examples

The following example illustrates how the calls might be used to invoke another NetView for AIX application on a list of objects.

```
/*  If OK  */
if (OVwCheckAction("Node Configuration Application",
                   "Configure", selections, mapInfo, submapId) < 0) {
      fprintf(stderr, "Failure: %s\n", OVwErrorMsg(OVwError()));
      return –1;
}
/*  Do It  */
else if (OVwDoAction("Node Configuration Application",
                   "Configure", selections, mapInfo, submapId) < 0) {
      fprintf(stderr, "Failure: %s\n", OVwErrorMsg(OVwError()));
      return –1;
}
```

Implementation Specifics

OVwCheckAction supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwCheckAction or OVwDoAction, link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwConfirmAcknowledgeObjectsCB(3)

Purpose

Functions as a callback for an acknowledge or unacknowledge object event

Related Functions

OVwConfirmUnacknowledgeObjectsCB

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmAcknowledgeObjectsCB) (void *userData, OVwEvntType type,
       OVwMapInfo *map, OVwObjectList *objectList);

void (*OVwConfirmUnacknowledgeObjectsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);
```

Description

OVwConfirmAcknowledgeObjectsCB and OVwConfirmUnacknowledgeObjectsCB are invoked in applications that have registered them whenever an acknowledge or unacknowledge operation is selected by the user. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when an object is acknowledged or unacknowledged should register these callbacks by using the OVwAddCallback function call, using ovwConfirmAcknowledgeObjects and ovwConfirmUnacknowledgeObjects as the event types.

The scope of the acknowledge and the unacknowledge operation is the open map. If the user selects an object on a map and then acknowledges that object, all symbols for that object on that map are acknowledged. But the operation does not cross maps, so symbols for that object in other maps will remain unacknowledged. The same is true for the unacknowledge operation.

The NetView for AIX program will send an ovwConfirmAcknowledgeObjects or ovwConfirmUnacknowledgeObjects event each time the user acknowledges or unacknowledges an object.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to a list of objects that have been acknowledged or unacknowledged.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmAcknowledgeObjects or ovwConfirmUnacknowledgeObjects. This is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Examples

The following example illustrates how to register to receive notifications from managed operations:

```
void
acknowledgeObjectsProc(char *userData, OVwEventType type,
                  OVwMapInfo *map, OVwObjectList *objectList)
{
  /* process notification here */
}

OVwAddCallback(ovwConfirmAcknowledgeObjects, NULL,
    (OVwCallbackProc) manageObjectsProc, NULL);
```

Implementation Specifics

OVwConfirmAcknowledgeObjectsCB and OVwConfirmUnacknowledgeObjectsCB support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmAcknowledgeObjectsCB or OVwConfirmUnacknowledgeObjectsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmCapabilityChangeCB(3)

Purpose

Functions as a callback for an object capability-change event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmCapabilityChangeCB) (void *userData,
       OVwEventType type, OVwMapInfo *map, OVwObjectList *objectList);
```

Description

OVwConfirmCapabilityChangeCB handles events sent to applications that have registered to receive them when new capability fields have been set for an object. These fields are set for an object when the symbol type is set for a symbol representing the object. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

To receive an event indicating that new capability fields have been set for an object, use OVwAddCallback to register a callback function of type OVwConfirmCapabilityChangeCB to be called when an ovwConfirmCapabilityChange event is generated.

In a symbol type registration file, it is possible to define default capability-field values for a symbol type. These capability-field values can be automatically set for an object when a symbol of the object has the given symbol type. Default capabilities for an object can be set based on symbol type in the following ways:

- Calling OVwCreateSymbol or one of its related functions with the ovwMergeDefaultCapabilities flag set
- Calling OVwSetSymbolType with the ovwMergeDefaultCapabilities flag set
- Changing the symbol type of a symbol by using the graphical interface

In these cases, the existing field values of an object are not changed. A default field value is set for an object only if the object currently has no value set for that field.

The capability-change event enables an application to verify that a capability field value, set for an object based on the symbol type, is valid for a particular object. The field_values field of OVwObjectInfo structures returned for each object by this event contains a complete list of the capability fields set for the object, including an indication of which new field values have been set (the modified flag of the OVwFieldValue structure).

Note: The ovwConfirmCapabilityChange event provides notification of capability field changes based on only setting the symbol type; it is not a general event for all capability-field changes. Capability field values can also change through the Add Object editing operation (see “OVwVerifyAdd(3)” on page 833), the Describe Object editing operation (see “OVwVerifyDescribeChange(3)” on page 849), and direct updates (see “OVwDbSetFieldValue(3)” on page 676). If capability-field values are changed by another application using OVwDbSetFieldValue, no notification event is generated.

Parameters

map
Specifies a pointer to a MapInfo structure for the open map on which the event occurred. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to a list of objects with new capability fields set.

type
Specifies the type of event that caused this callback to be invoked, namely ovwConfirmCapabilityChange. This parameter is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmCapabilityChangeCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmCapabilityChangeCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwSetSymbolType(3)” on page 822.
- See “OVwVerifyAdd(3)” on page 833.
- See “OVwVerifyDescribeChange(3)” on page 849.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwConfirmCreateObjectsCB(3)

Purpose

Functions as a callback for a create-object event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmCreateObjectsCB) (void *userData,
       OVwEventType type, OVwMapInfo *map, OVwObjectList *objectList);
```

Description

OVwConfirmCreateObjectsCB handles events sent to applications that have registered to receive them when an object is created on the open map. An object is created on a map when the first map symbol representing the object is created or when a submap that has a parent object, which is not yet represented by a symbol on the map, is created. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when an object is created on the open map must register for this callback, using the OVwAddCallback function call and using ovwConfirmCreateObjects as the event type.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to the list of created objects.

type
Specifies the event which invoked the callback, namely ovwConfirmCreateObjects. This field is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback

Implementation Specifics

OVwConfirmCreateObjectCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmCreateObjectsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmCreateSubmapsCB(3)

Purpose

Functions as a callback for a create-submap event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmCreateSubmapsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSubmapList *submapList);
```

Description

OVwConfirmCreateSubmapsCB manages events sent to applications that have registered to receive them when a submap is created on the open map. The NetView for AIX program will generate the ovwConfirmCreateSubmaps event when the user or an application creates a submap. See "OVwApiIntro(5)" on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

For an application to be notified when a submap is created on the open map, the application should register for this callback by using the OVwAddCallback function call and ovwConfirmCreateSubmaps as the event type.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submapList
Specifies a pointer to the list of created submaps.

type
Specifies the event that invoked the callback, namely ovwConfirmCreateSubmaps. This field is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmCreateSubmapsCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmCreateSubmapsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmCreateSymbolsCB(3)

Purpose

Functions as a callback for a create symbol event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmCreateSymbolsCB) (void *userData, OVwEventType type, OVwMapInfo *map,
       OVwSymbolList *symbolList);
```

Description

OVwConfirmCreateSymbolsCB handles events sent to applications that have registered to receive them when a symbol is created on the open map. The NetView for AIX program will generate the ovwConfirmCreateSymbols event when the user or an application creates a symbol. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

For an application to be notified when a symbol is created on the open map, the application should register for this callback by using the OVwAddCallback function call and ovwConfirmCreateSymbols as the event type.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map

symbolList
Specifies a pointer to the list of created symbols.

type
Specifies the event that invoked the callback, namely ovwConfirmCreateSymbols. This field is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmCreateSymbolsCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmCreateSymbolsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmDeleteObjectsCB(3)

Purpose

Functions as a callback for a delete-object event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmDeleteObjectsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);
```

Description

OVwConfirmDeleteObjectsCB manages events sent to applications that have registered to receive them when an object is deleted from the map. An object is deleted from the map when the last symbol of the object is deleted. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API, including the role of the asynchronous NetView for AIX events.

For an application to be notified when a delete operation is selected by the user or another application, the application should register for this callback, using the OVwAddCallback function call and ovwConfirmDeleteObjects as the callback type.

This delete event is generated as a result of the user or another application deleting symbols or submaps from the map.

If the last symbol for a given object is deleted, the graphical interface deletes that object and sends an ovwConfirmDeleteObjects event. For an ovwConfirmDeleteObjects event, there is no ovwQueryDeleteObject event. This is because applications do not have the opportunity to reject an object-delete operation. When an object is deleted from a map, the graphical interface sends an ovwConfirmDeleteObject event to applications that are registered to receive it. The OVwConfirmDeleteObjectsCB routine must then check the op_scope field of the OVwObjectInfo structure to determine the scope of the operation. That field can have one of two values: ovwOpenMapScope or ovwAllMapsScope. In the case of ovwOpenMapScope, the callback routine needs to update its current structures to reflect this delete. For ovwAllMapsScope, however, it must delete that object from its database.

The application must ensure that the object is removed from the object database. If the op_scope field is ovwAllMapsScope, the application must unset all the fields of the object that it set and then delete the object from the object database. See “OVwDbDeleteObject(3)” on page 641 for more details.

How the OVwConfirmDeleteObjectsCB manages this event depends on the implementation of the application. If there is a central database serving multiple instances of the application (not necessarily simultaneously), the ovwAllMapsScope scope is an indicator that the object no longer resides on any map so it should be removed from that central database. The ovwOpenMapScope means that the object is still on another map and there is the possibility that another instance of this application will still need that object to reside in the database.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to a list of the objects deleted.

type
Specifies the type of event that caused the callback to be invoked, namely ovwConfirmDeleteObjects. This field is useful if one callback handles multiple events. This field is useful if one callback handles multiple event types.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmDeleteObjectsCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmDeleteObjectsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See OVwConfirmDeleteSymbolsCB in “OVwVerifyDeleteSymbol(3)” on page 846.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwConfirmDeleteSubmapsCB(3)

Purpose

Functions as a callback for a delete submap event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmDeleteSubmapsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSubmapList *submapList);
```

Description

OVwConfirmDeleteSubmapsCB handles events sent to applications that have registered to receive them when an submap is deleted from the map. An submap is deleted from the map when the last symbol of the submap is deleted. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

For an application to be notified when a submap is deleted from the open map, the application should register for this callback, using the OVwAddCallback function call, to use ovwConfirmDeleteSubmaps as the event type.

The NetView for AIX program sends this event when the user or an application deletes a submap.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submapList
Specifies a pointer to the list of deleted submaps.

type
Specifies the event type that invoked the callback, namely ovwConfirmDeleteSubmaps. This field is useful if one callback manages multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmDeleteSubmapsCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmDeleteSubmapsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmExplodeObjectCB(3)

Purpose

Functions as a callback for an explode object event

Syntax

```
#include <OV/ovw.h>

    void (*OVwConfirmExplodeObjectCB) (void *userData, OVwEventType type,
           OVwMapInfo *map, OVwObjectInfo *object);
```

Description

OVwConfirmExplodeObjectCB is invoked in applications that have registered it whenever the user explodes an object. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when an object is exploded should register this callback by using the OVwAddCallback function call, using ovwConfirmExplodeObject as the event type.

An object is exploded by the user when one of its symbols is either dragged or double clicked to show the child submap associated with that object. The event is sent to the applications even if the object does not have a child submap or when the child submap is already opened or displayed.

The event will not be sent to the applications if the user is acting on a executable symbol or if the symbol is in a map snapshot.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

object
Specifies a pointer to the structure representing the object being exploded.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmExplodeObject. This is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Examples

The following example illustrates how to register to receive notifications from managed operations:

```
void
explodeObjectProc(char *userData, OVwEventType type,
                  OVwMapInfo *map, OVwObjectInfo *object)
{
  /* process notification here */
}

OVwAddCallback(ovwConfirmExplodeObject, NULL,
    (OVwCallbackProc) explodeObjectProc, NULL);
```

Implementation Specifics

OVwConfirmExplodeObjectCB support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmExplodeObjectCB link to the following libraries:

/usr/OV/lib/libovw.a

/usr/OV/lib/libov.a

/usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmHideSymbolsCB(3)

Purpose

Functions as a callback for a hide symbol event

Related Functions

OVwConfirmUnhideSymbolsCB

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmHideSymbolsCB) (void *userData*, OVwEventType type,
       OVwMapInfo *map, OVwSymbolList *symbolList);

void (*OVwConfirmUnhideSymbolsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolList *symbolList);
```

Description

When a symbol is hidden or unhidden, OVwConfirmHideSymbolsCB and OVwConfirmUnhideSymbolsCB manage events sent to applications that have been registered to receive them. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

For an application to be notified when an object is hidden or unhidden, the application must be registered for these callbacks through the OVwAddCallback function call, using ovwConfirmHideSymbols and ovwConfirmUnhideSymbols as the event types.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbolIdList
Specifies a pointer to a list of symbol IDs that have been hidden or unhidden.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmHideSymbols or ovwConfirmUnhideSymbols. This is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Examples

- The following is an example of registering to receive notifications of hide operations.

```
void
hideSymbolsProc(char *userData, OVwEventType type,
                OVwSymbolIdList *symbolIdList)
{
   /* process notification here */
}

OVwAddCallback(ovwConfirmHideSymbols, NULL,
               (OVwCallbackProc) hideSymbolsProc, NULL);
```

Implementation Specifics

OVwConfirmHideSymbolsCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmHideSymbolsCB or OVwConfirmUnhideSymbolsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmManageObjectsCB(3)

Purpose

Functions as a callback for a manage object event

Related Functions

OVwConfirmUnmanageObjectsCB

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmManageObjectsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);

void (*OVwConfirmUnmanageObjectsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);
```

Description

OVwConfirmManageObjectsCB and OVwConfirmUnmanageObjectsCB are invoked in applications that have registered them whenever a manage or unmanage operation is selected by the user. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when an object is managed or unmanaged should register these callbacks by using the OVwAddCallback function call, using ovwConfirmManageObjects and ovwConfirmUnmanageObjects as the event types.

The scope of the manage and the unmanage operation is the open map. If the user selects an object on a map and then manages that object, all symbols for that object on that map are managed. But the operation does not cross maps, so symbols for that object in other maps will remain unmanaged. The same is true for the unmanage operation.

The NetView for AIX program will send an ovwConfirmManageObjects or ovwConfirmUnmanageObjects event each time the user manages or unmanages an object. For example, when the user performs an unmanage operation, the op_scope field of the OVwObjectInfo structure is set to either ovwOpenMapScope or ovwAllMapsScope, to indicate the scope of the results of the operation. If the user unmanages an object in the open map and that object is still managed in another map, the op_scope field is set to ovwOpenMapScope. In this case, the unmanage operation applies to only the open map.

In another example, if the user unmanages an object in the open map and the object is now unmanaged in all maps, the op_scope field is set to ovwAllMapsScope. In this case, the unmanage operation applies to all maps. Although the manage and unmanage operations apply only to the open map, if the operation causes an object to be managed or unmanaged in the last map, the scope of the operation is said to be across all maps.

These events are typically used to configure the discovery process. The discovery process is the part of the application that discovers and monitors network objects. Your application may not have discovery and monitoring capability, in which case these events are not needed. When no maps have a particular object managed, then the discovery process does not need to query that object. But, if at least one map has

that object managed, the discovery process is required to query that object so that status events, as well as others, can be generated.

The manage and unmanage operations are recursive user interface operations. If a compound object is managed or unmanaged, the NetView for AIX program will traverse all submaps below that object and manage or unmanage any objects found in those submaps that do not have a symbol in the submap that contains the selected compound object. Each object that is managed or unmanaged will be added to the objectList.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to a list of objects that have been managed or unmanaged.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmManageObjects or ovwConfirmUnmanageObjects. This is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Examples

The following example illustrates how to register to receive notifications from managed operations:

```
void
manageObjectsProc(char *userData, OVwEventType type,
                  OVwMapInfo *map, OVwObjectList *objectList)
{
  /* process notification here */
}

OVwAddCallback(ovwConfirmManageObjects, NULL,
    (OVwCallbackProc) manageObjectsProc, NULL);
```

Implementation Specifics

OVwConfirmManageObjectsCB and OVwConfirmUnmanageObjectsCB support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmManageObjectsCB or OVwConfirmUnmanageObjectsCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmMoveSymbolCB(3)

Purpose

Functions as a callback for a move-symbol event

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmMoveSymbolCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolInfo *symbol);
```

Description

When a move operation is selected by the user, OVwConfirmMoveSymbolCB manages events sent to applications that have registered to receive them. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

For an application to be notified when a symbol is moved, the application should register this callback, using the OVwAddCallback function call with ovwConfirmMove as the callback type.

When the user moves a symbol on a submap, the NetView for AIX program will send an ovwConfirmMove event to the applications. The applications can use the new position as valid semantic data or strictly presentation data. The OVwSymbolInfo structure contains symbol position information.

The ovwConfirmMoveSymbol event is also generated when an application makes an OVwSetSymbolPosition call. In that case, the position information might not indicate the requested move, depending on whether the submap is displayed and automatic layout is enabled. See “OVwSetSymbolPosition(3)” on page 815 for more details.

The ovwConfirmMoveSymbol event is not sent to applications when the user moves a symbol from one submap to another; the NetView for AIX program will treat the cut-and-paste operations like symbol delete-and-add operations.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbol
Specifies a pointer to the structure representing the symbol being moved.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmMoveSymbol. This is useful if one callback handles multiple events.

userData
Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwConfirmMoveSymbolCB supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmMoveSymbolCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwSetSymbolPosition(3)” on page 815.
- See “OVwApiIntro(5)” on page 560.

OVwConfirmObjectStatusCB(3)

Purpose

Functions as a callback for a change objects status event

Related Functions

OVwConfirmSymbolStatusCB
OVwConfirmCompoundStatusCB

Syntax

```
#include <OV/ovw.h>

void (*OVwConfirmObjectStatusCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);

void (*OVwConfirmSymbolStatusCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolList *symbolList);

void (*OVwConfirmCompoundStatusCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectList *objectList);
```

Description

To receive an event indicating one of these status changes, use OVwAddCallback to register a callback function using ovwConfirmObjectStatusChange, ovwConfirmSymbolStatusChange or ovwConfirmCompoundStatusChange as the event type.

The ovwConfirmObjectStatusChange event is generated when an application makes a call to OVwSetStatusOnObject that changes the status of an object or when the user initiates a manage, unmanage, acknowledge, or unacknowledge operation.

The ovwConfirmCompoundStatusChange event is generated when the compound-status of an object is changed. The compound-status is the status propagated to the object from symbols in its child submap according to the selected propagation rules. The compound status of the parent object changes as a result of one of the symbols in its child submap changing status. When the symbol in the child submap changes status, the compound status of the parent object is recomputed and displayed as appropriate.

The ovwConfirmSymbolStatusChange event is generated when the status of a symbol is changed. This can result from a change in the object status of its associated object, a change in the compound status of its associated object, or a direct change in the symbol's status through OVwSetStatusOnSymbol.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

objectList
Specifies a pointer to the list of objects that changed status.

symbolList
Specifies a pointer to the list of symbols that changed status.

type
Specifies the type of NetView for AIX event that caused this callback to be invoked, namely ovwConfirmObjectStatus, ovwConfirmSymbolStatus, or ovwConfirmCompoundStatus. This is useful if one callback manages multiple, event types.

userData
Specifies a pointer to the user data registered for the callback

Implementation Specifics

OVwConfirmObjectStatusCB and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwConfirmObjectStatusCB or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwSetStatusOnObject(3)” on page 803.
- See OVwSetStatusOnSymbols in “OVwSetStatusOnObject(3)” on page 803.
- See “OVwSetSymbolStatusSource(3)” on page 820.
- See “OVwApiIntro(5)” on page 560.

OVwCreateAction(3)

Purpose

Manipulates application action registration

Related Functions

OVwDeleteAction
OVwGetAction
OVwSetAction
OVwFreeActionRegInfo

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwCreateAction(char *actionId, OVwActionRegInfo *actionInfo);

int OVwDeleteAction(char *actionId);

int OVwSetAction(char *actionId, OVwActionRegInfo *actionInfo);

OVwActionRegInfo *OVwGetAction(char *actionId);

void OVwFreeActionRegInfo(OVwActionRegInfo *actionInfo);
```

Description

OVwCreateAction creates the specified action in the current registration context.

OVwDeleteAction deletes the specified action in the current registration context.

OVwGetAction retrieves registration information for the specified action in the current registration context.

OVwSetAction modifies registration information for the specified action in the current registration context.

OVwFreeActionRegInfo frees the memory allocated for an OVwActionRegInfo structure. It should be used to free the OVwActionRegInfo structure returned by OVwGetAction when it is no longer needed.

Before calling OVwCreateAction, OVwDeleteAction, or OVwSetAction, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the action registration become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to create action registration dynamically. If your application's action registration is static, use the application registration files to define application actions and their menu structure.

Parameters

actionId
Specifies a pointer to the name of the action as defined in the application registration file for the current registration context.

actionInfo

Specifies a pointer to an OVwActionRegInfo structure. The OVwActionRegInfo structure contains the elements of the action registration information. In <OV/ovw_reg.h>:, it is defined as shown in the following example:

```
typedef struct {
    char *selection_rule;
    int min_selected;
    int max_selected;
    char *command;
    unsigned long process_flags;
    OVwFieldBindList *name_fields;
    char *callback_args;
} OVwActionRegInfo;
```

The members of this structure are:

selection_rule Specifies the action selection rule as contained in the SelectionRule statement within action declaration of the application registration file for the current registration context. For no SelectionRule, set this field to NULL.

min_selected Specifies the value for the MinSelected statement within the action declaration. For no MinSelected statement, set this field to ovwDefaultActionMinSelected, which is defined in <OV/ovw_reg.h>.

max_selected Specifies the value for the MaxSelected statement within the action declaration. For no MaxSelected statement, set this field to ovwDefaultActionMaxSelected, which is defined in <OV/ovw_reg.h>.

command Specifies the command string for the Command statement within the action declaration. For no command statement, set this field to NULL.

process_flags Specifies the options for the action command statement within the action declaration of the application registration file for the current registration context. The value is a mask that is the logical OR of the following constants from <OV/ovw_reg.h>

ovwProcInitial — Starts the application when the NetView for AIX program starts.

ovwProcShared — Specifies that one instance of the application can handle multiple action requests.

ovwProcRestart — Restarts the application automatically when the application dies or exits. This constant should be used for applications which manage maps and which should run for the duration of an NetView for AIX session.

name_fields Specifies an OVwFieldBindList containing the field IDs for those fields which should be used in the NameField statement within the action declaration. The field values (the field_val field) of the OVwFieldBinding members are ignored and should be set to NULL. For no NameField statement, set this field to NULL.

callback_args Specifies the string which should be used for the CallbackArgs statement within the action declaration. For no CallbackArgs statement, set this field to NULL.

Return Values

If successful, OVwCreateAction, OVwDeleteAction, and OVwSetAction return 0 (zero). If unsuccessful, they return –1 (negative one).

If successful, OVwGetAction returns a pointer to an OVwActionRegInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwCreateAction, OVwDeleteAction, OVwGetAction, and OVwSetAction set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_EXISTS]	The specified actionId is already registered in the current registration context.
[OVw_ACTION_MINSELECTED_ERROR]	The specified min_selected value is not valid.
[OVw_ACTION_MAXSELECTED_ERROR]	The specified max_selected value is either not valid or conflicts with the min_selected setting.
[OVw_ACTION_NOT_FOUND]	The specified actionId is not registered in the current registration context.
[OVw_ACTION_SELECTION_RULE_ERROR]	There is a syntax error or a semantic error in the specified selection_rule.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	There is not enough memory to store the callback registration information.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been started with OVwInit.
[OVw_PERMISSION_DENIED]	OwLockRegUpdates was not called prior to calling this function.

Implementation Specifics

OVwActionRegistration supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwActionRegistration functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwCreateApp(3)

Purpose

Manipulates NetView for AIX application registration information

Related Functions

OVwDeleteApp
OVwGetApp
OVwSetApp
OVwFreeAppRegInfo

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwCreateApp(char *appName, OVwAppRegInfo *appInfo);

int OVwDeleteApp(char *appName);

int OVwSetApp(OVwAppRegInfo *appInfo);

OVwAppRegInfo *OVwGetApp();

void OVwFreeAppRegInfo(OVwAppRegInfo *appInfo);
```

Description

OVwCreateApp creates the specified ovw application by creating registration information for it.

OVwDeleteApp deletes the specified ovw application registration.

OVwGetApp retrieves registration information for the application that is the current registration context.

OVwSetApp modifies registration information for the application that is the current registration context.

OVwFreeAppRegInfo frees the memory allocated for an OVwAppRegInfo structure. It should be used to free the OVwAppRegInfo structure returned by OVwGetApp when it is no longer needed.

Before calling OVwCreateApp, OVwDeleteApp, or OVwSetApp, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the application registration become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to create or modify application registration dynamically. If your application registration is static, use the application registration files for defining registration information.

Parameters

appName
Specifies a pointer to the name of an NetView for AIX application.

appInfo
Specifies a pointer to an OVwAppRegInfo structure. The OVwAppRegInfo structure contains certain global elements of the application registration information. It is defined in <OV/ovw_reg.h> as follows:

```
typedef struct {
   char *parent_name;
   char *reg_file;
   char *command;
   unsigned long process_flags;
   char **description;
   char **copyright;
   char *version;
   char *help_directory;
   OVwFieldBindList *name_fields;
} OVwAppRegInfo;
```

The members of this structure are:

parent_name
: The name of the parent application for the application. If the application has no parent, set this field to NULL.

reg_file
: The name of the application registration file. If you wish to save the registration information, you must specify a value for this field. If the specified file name is a relative path, it is assumed to be relative to the NetView for AIX application registration directory.

command
: The command string for the application Command statement within the application block of the registration file. If you do not want a Command statement. set this field to NULL.

process_flags
: The options for the application Command statement in the application registration file for the current registration context. The value is a mask that is a logical OR of the following constants from <OV/ovw_reg.h>:

  ovwProcInitial
  : Starts the application when the NetView for AIX program starts.

  ovwProcShared
  : Specifies that one instance of the application can handle multiple action requests.

  ovwProcRestart
  : Restarts the application automatically if it ever terminates or exits. This should be used for applications that manage maps and that should run for the duration of an NetView for AIX session.

description
: A NULL-terminated array of strings which are those listed in the application Description statement in the application registration file. If you do not want a Description statement, set this field to NULL.

copyright
: A NULL-terminated array of strings which are those listed in the application Copyright statement in the application registration file. If you do not want a Copyright statement, set this field to NULL.

help_directory	The string specified for the application HelpDirectory statement in the application registration file. If you do not want a HelpDirectory statement, set this field to NULL.
version	The string specified for the application Version statement in the application registration file. If you do not want a Version statement, set this field to NULL.
name_fields	An OVwFieldBindList containing the field IDs for those fields which should be used in the NameField block of the application registration file. The field values (the field_val field) of the OVwFieldBinding members are ignored, and should be set to NULL. If you do not want a NameField statement, set this field to NULL.

Return Values

If successful, OVwCreateApp, OVwDeleteApp, and OVwSetApp return 0 (zero). If unsuccessful, they return –1 (negative one).

If successful, OVwGetApp returns a pointer to an OVwAppRegInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwCreateApp, OVwDeleteApp, OVwGetApp, and OVwSetApp set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_EXISTS]	The specified appName is already a registered application.
[OVw_APP_NOT_FOUND]	The specified appName is not a registered application.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	There is not enough memory to store the callback registration information.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	You have not called OVwLockRegUpdates prior to calling this function.

Implementation Specifics

OVwAppRegistration supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwAppRegistration functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwCreateMenu(3)

Purpose

Creates and deletes menu registration

Related Functions

OVwDeleteMenu

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwCreateMenu(char *menuId);

int OVwDeleteMenu(char *menuId);
```

Description

OVwCreateMenu creates a menu or object menu in the current registration context. Once a menu is created, menu items can be added to it with OVwAddMenuItem.

OVwDeleteMenu deletes the specified menu from the current registration context. The specified menu must not contain any menu items and must not be referenced by other menu items.

Before calling these functions, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu item registration only become permanent after calling OVwSaveRegUpdates.

Use these functions if your application needs to create menu registration dynamically. If your application menu registration is static, use the application registration files to define the application menu structure.

Parameters

menuId
Specifies a pointer to the identifier of the menu as it appears or will appear in the application registration file of the current registration context.

Return Values

If successful, OVwCreateMenu and OVwDeleteMenu return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwCreateMenu and OVwDeleteMenu set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENU_EXISTS]	The specified menuId is already registered in the current registration context.

[OVw_MENU_IN_USE]	The specified menu menuId is still referenced by other menu items and cannot be deleted.
[OVw_MENU_NOT_EMPTY]	The menu still contains menu items and cannot be deleted.
[OVw_MENU_NOT_FOUND]	The specified menuId is not registered in the current registration context.
[OVw_OUT_OF_MEMORY]	There is not enough memory to store the callback registration information.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	You have not called OVwLockRegUpdates prior to calling this function.

Implementation Specifics

OVwMenuItemRegistration and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwMenuItemRegistration function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwCreateMenuItem(3)

Purpose

Manipulates menu item registration information

Related Functions

OVwDeleteMenuItem
OVwGetMenuItem
OVwSetMenuItem
OVwFreeMenuItemRegInfo

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwCreateMenuItem(OVwMenuItemRegInfo *menuItemInfo);

int OVwDeleteMenuItem(char *menuItemId);

int OVwSetMenuItem(char *menuItemId, OVwMenuItemRegInfo *menuItemInfo);

OVwMenuItemRegInfo *OVwGetMenuItem(char *menuItemId);

void OVwFreeMenuItemRegInfo(OVwMenuItemRegInfo *menuItemInfo);
```

Description

OVwCreateMenuItem creates a menu item in the current registration context. (For creating an object menu item, see “OVwCreateObjMenuItem(3)” on page 616).

OVwDeleteMenuItem deletes the specified menu item from the current registration context. The specified menu item must not be included in any menu and must not contain any menu item functions.

OVwGetMenuItem retrieves registration information for the specified menu item in the current registration context.

OVwSetMenuItem modifies registration information for the specified menu item in the current registration context.

OVwFreeMenuItemRegInfo frees the memory allocated for an OVwMenuItemRegInfo structure. It should be used to free the OVwMenuItemRegInfo structure returned by OVwGetMenuItem when it is no longer needed.

Before calling OVwCreateMenuItem, OVwDeleteMenuItem, or OVwSetMenuItem, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu item registration become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to create menu registration dynamically. If your application menu registration is static, use the application registration files to define the application menu structure.

Parameters

menuItemId
Specifies a pointer to a menu item identifier returned from OVwCreateMenuItem or from OVwFindMenuItem.

menuItemInfo
Specifies a pointer to an OVwMenuItemRegInfo structure. The OVwMenuItemRegInfo structure contains the elements of the registration information for an application menu item. It is defined as follows in the <OV/ovw_reg.h> header file:

```
typedef struct {
     char *label;
     char *mnemonic;
     char *accelerator;
     int precedence;
} OVwMenuItemRegInfo;
```

The members of this structure are:

label	A pointer to the menu item label string.
mnemonic	A pointer to a string specifying the mnemonic for the menu item. The first character in the string is used as the mnemonic. If no mnemonic is specified for the menu item, this field is NULL.
accelerator	A pointer to a string specifying the accelerator key sequence for the menu item. If no accelerator is specified for the menu item, this field is NULL.
precedence	The precedence value for the menu item. Precedence values may be within the range from and including ovwMinMenuItemPrecedence (defined as 0 in <OV/ovw_reg.h>) and ovwMaxMenuItemPrecedence (defined as 100 in <OV/ovw_reg.h>). If no specific precedence is needed for the menu item, this field should be set to ovwDefaultMenuItemPrecedence (defined as 50 in <OV/ovw_reg.h>).

Return Values

If successful, OVwCreateMenuItem, OVwDeleteMenuItem, and OVwSetMenuItem return 0 (zero). If unsuccessful, they return –1 (negative one). Because the return value for OVwCreateMenuItem is dynamically allocated, you must free the string when it is no longer needed.

If successful, OVwGetMenuItem returns a pointer to an OVwMenuItemRegInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwCreateMenuItem and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENUITEM_EXISTS]	The specified menuItemId is already registered in the current registration context.
[OVw_MENUITEM_IN_USE]	The specified menu item, menuId, is still included in a menu and cannot be deleted.
[OVw_MENUITEM_NOT_EMPTY]	The menu item still contains menu item functions and cannot be deleted.

[OVw_MENUITEM_NOT_FOUND]	The specified menuItemId is not registered in the current registration context.
[OVw_OUT_OF_MEMORY]	There is not enough memory to store the callback registration information.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	You have not called OVwLockRegUpdates prior to calling this function.
[OVw_MENUITEM_PRECEDENCE_ERROR]	The specified precedence is not within the valid range of precedence values.

Implementation Specifics

OVwMenuItem and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwMenuItem function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwCreateObjMenuItem(3)

Purpose

Manipulates object menu item registration information

Related Functions

OVwDeleteObjMenuItem
OVwGetObjMenuItem
OVwSetObjMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwCreateObjMenuItem(OVwMenuItemRegInfo *menuItemInfo);

int OVwSetObjMenuItem(char *objMenuItemId, OVwMenuItemRegInfo *objMenuItemInfo);

OVwMenuItemRegInfo *OVwGetObjMenuItem(char *objMenuItemId);
```

Description

OVwCreateObjMenuItem creates a menu item in the current registration context to be incorporated in the Object Menu structure.

OVwDeleteObjMenuItem deletes registration information for the specified Object Menu item in the current registration context.

OVwGetObjMenuItem retrieves registration information for the specified Object Menu item in the current registration context.

OVwSetObjMenuItem modifies registration information for the specified Object Menu item in the current registration context.

Before calling OVwCreateObjMenuItem, OVwGetObjMenuItem, or OVwSetObjMenuItem, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the menu item registration become permanent only after calling OVwSaveRegUpdates.

Use these functions if your application needs to create Object Menu items dynamically. If your application menu registration is static, use the application registration files to define the application menu structure.

Parameters

menuItemInfo

Specifies a pointer to an OVwMenuItemRegInfo structure. The OVwMenuItemRegInfo structure contains the elements of the registration information for an application menu item. The structure is defined as follows in the <OV/ovw_reg.h> header file:

```
typedef struct {
     char *label;
     char *mnemonic;
     char *accelerator;
     int  precedence;
} OVwMenuItemRegInfo;
```

The members of this structure are:

label — A pointer to the menu item label string.

mnemonic — A pointer to a string specifying the mnemonic for the menu item. The first character in the string is used as the mnemonic. If no mnemonic is specified for the menu item, this field is NULL.

accelerator — A pointer to a string specifying the accelerator key sequence for the menu item. If no accelerator is specified for the menu item, this field is NULL.

precedence — The precedence value for the menu item. Precedence values can be within the range from and including ovwMinMenuItemPrecedence (defined as 0 in <OV/ovw_reg.h>) and ovwMaxMenuItemPrecedence (defined as 100 in <OV/ovw_reg.h>). If no specific precedence is needed for the menu item, set this field to ovwDefaultMenuItemPrecedence (defined as 50 in <OV/ovw_reg.h>).

Return Values

If successful, OVwCreateMenuItem returns a pointer to an OVwMenuItemRegInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwCreateObjMenuItem sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] — The connection to the NetView for AIX program was lost.

[OVw_MENUITEM_EXISTS] — The specified menuItemId is already registered in the current registration context.

[OVw_OUT_OF_MEMORY] — There is not enough memory to store the callback registration information.

[OVw_OVW_NOT_INITIALIZED] — The EUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED] — You have not called OVwLockRegUpdates prior to calling this function.

[OVw_MENUITEM_PRECEDENCE_ERROR] — The specified precedence is not within the valid range of precedence values.

Implementation Specifics

OVwCreateObjMenuItem supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwCreateObjMenuItem function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwSaveRegUpdates(3)” on page 797.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwCreateSubmap(3)

Purpose

Creates a submap

Related Functions

OVwDeleteSubmap

Syntax

```
#include <OV/ovw.h>

OVwSubmapId OVwCreateSubmap(OVwMapInfo *map, OVwObjectId parentObject,
       int submapPolicy, int submapType, char *submapName,
       int layout, unsigned int flags);

int OVwDeleteSubmap(OVwMapInfo *map, OVwSubmapId submapId);
```

Description

OVwCreateSubmap creates a new submap on the open map. If the argument parentObject is a valid object ID, the submap is created as the child submap of the specified object. If the argument parentObject is ovwNullObjectId, the submap is created as an orphan submap and has no parent object.

All submaps have two planes on which symbols can appear, a user plane and an application plane. Symbols created by an application is placed on the application plane. A symbol added by a user can be placed on either the user plane or the application plane, depending on whether it is accepted by an application (see “OVwVerifyAdd(3)” on page 833). A user can create and delete submaps.

A submap is created with a policy that specifies who has permission to modify it. A submap can be either exclusive or shared. If a submap is exclusive, the only application that can update the submap is its creator; the creator has exclusive control over the application plane of the submap. If a submap is shared, any application can update the submap. Updating a submap includes creating and deleting symbols, setting symbol and submap attributes, and deleting the submap.

Applications may restrict the operations the user is able to perform over a submap. This is done by creating the submap with read-only permission so that the user cannot update the submap. If the submap is shared, other applications can still update the submap, even if it is read-only. A read-only submap behaves much like any submap in a read-only map.

It is recommended that applications create shared submaps as much as possible, rather than exclusive submaps, to enable greater cooperation among applications. Submaps created by a user through the graphical interface are always shared. In a shared submap, it is possible for an application to mark the subset of symbols in which it is interested. This is done by creating a symbol or using OVwSetSymbolApp to express interest in a symbol created by another application. For more information about creating a symbol, see “OVwCreateSymbol(3)” on page 623.

Creating a submap does not cause the submap to be displayed. Because the user can display submaps through the graphical interface, most applications do not need to issue a call to display a submap. A submap that is created as a result of an interactive user request, for example, a menu operation, can be displayed by an application. Submaps can be displayed with the routine OVwDisplaySubmap.

OVwDeleteSubmap deletes a submap from the open map. The only application that can delete an exclusive submap is the one that created it. Any application can delete a shared submap.

Deleting a submap deletes all symbols on the submap but does not delete the parent object. Deletion is a recursive operation. If the last symbol of an object is deleted, the object associated with the symbol is deleted from the map. If the deleted object has a child submap, the child submap and all its symbols are deleted, and so on. The following list provides the events that can be used to determine what was deleted, and their related callbacks:

ovwConfirmDeleteSymbols
: OVwConfirmDeleteSymbolsCB

ovwConfirmDeleteSubmaps
: OVwConfirmDeleteSubmapsCB

ovwConfirmDeleteObjects OVwConfirmDeleteObjectsCB

Parameters

flags Specifies submap creation flags. This is the logical OR of the following flags, which are defined in <OV/ovw.h>

ovwNoSubmapFlags	This value can be specified if no submap flags are needed.
ovwDisableAutoLayout	The submap is created with automatic layout initially disabled.
ovwReadOnlySubmap	The submap is created with read-only permission to the user.

layout Specifies the automatic layout algorithm used for symbol placement in the submap. The following permitted values are defined in the <OV/ovw.h> header file:

ovwNoLayout	No layout algorithm is used.
ovwPointToPointLayout	A layout of interconnected symbols.
ovwRowColumnLayout	A row/column layout.
ovwBusLayout	A bus layout with a bus backbone symbol.
ovwStarLayout	A star layout, allowing a star center to be specified.
ovwRingLayout	A ring layout with a ring backbone symbol.
ovwTreeLayout	A tree layout.
ovwMultipleConnLayout	A layout for a list of connections.

map Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

parentObject Specifies the object ID of the parent object of the submap being created. This argument may be ovwNullObjectId. If the object does not yet exist on the open map, it will be created.

submapId Specifies the submap ID of the submap to delete.

submapName Specifies a pointer to the name of the submap to be created.

submapPolicy Specifies the policy of the submap. The following permitted values are defined in the <OV/ovw.h> header file:

ovwSharedSubmap	Any application can update the submap.
ovwExclusiveSubmap	Only the creating application can update the submap.

submapType Specifies an application-specific submap type. This value can be used by the creating application to tag different types of submaps. If used for this purpose, a non-zero value should be specified; otherwise, ovwNoSubmapType may be specified.

Return Values

If successful, OVwCreateSubmap returns a valid submap ID. If unsuccessful, it returns ovwNullSubmapId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing submap IDs.

If successful, OVwDeleteSubmap returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwCreateSubmap and OVwDeleteSubmap sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open read-only.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

OVwCreateSubmap returns the following additional errors:

[OVw_OBJECT_NOT_FOUND]	The argument parentObject is not the object ID of an existing object.
[OVw_SUBMAP_EXISTS]	The object identified by parentObject already has a child submap on the open map.
[OVw_SUBMAP_INVALID_LAYOUT]	The argument layout has a value that is not valid.
[OVw_SUBMAP_INVALID_POLICY]	The argument submapPolicy has a value that is not valid.

OVwDeleteSubmap returns the following additional errors:

[OVw_SUBMAP_NOT_FOUND]	The argument submapId is not the submap ID of a submap that exists on the map.
[OVw_SUBMAP_PERMISSION_DENIED]	The submap cannot be deleted, because it was created as an exclusive submap by another application or it is the root submap.

Implementation Specifics

OVwCreateSubmap supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwCreateSubmap function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1)
- See “OVwConfirmDeleteObjectsCB(3)” on page 588
- See “OVwConfirmDeleteSubmapsCB(3)” on page 590
- See “OVwCreateSymbol(3)” on page 623
- See “OVwDisplaySubmap(3)” on page 683
- See “OVwError(3)” on page 688
- See “OVwGetMapInfo(3)” on page 719
- See “OVwInit(3)” on page 741
- See “OVwIsIdNull(3)” on page 743
- See “OVwSetSymbolApp(3)” on page 808
- See “OVwVerifyAdd(3)” on page 833
- See “OVwVerifyDeleteSymbol(3)” on page 846
- See “OVwApiIntro(5)” on page 560

OVwCreateSymbol(3)

Purpose

Creates symbols

Related Functions

OVwCreateSymbols
OVwCreateSymbolByName
OVwCreateSymbolBySelectionName
OVwCreateSymbolByHostname
OVwCreateComponentSymbol
OVwCreateComponentSymbolByName
OVwCreateConnSymbol
OVwCreateConnSymbolByName
OVwDeleteSymbol
OVwDeleteSymbols

Syntax

```
#include <OV/ovw.h>

OVwSymbolId OVwCreateSymbol(OVwMapInfo *map,
       OVwSubmapId submapId, OVwObjectId objectId,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
       OVwSymbolPosition *symbolPosition, unsigned int flags);

int OVwCreateSymbols(OVwMapInfo *map, OVwSymbolCreateList *symbolList);

OVwSymbolId OVwCreateSymbolByName(OVwMapInfo *map,
       OVwSubmapId submapId, OVwFieldBinding *name,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
       OVwSymbolPosition *symbolPosition, unsigned int flags);

OVwSymbolId OVwCreateSymbolBySelectionName(OVwMapInfo *map,
       OVwSubmapId submapId, char *selectionName,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
       OVwSymbolPosition *symbolPosition, unsigned int flags);

OVwSymbolId OVwCreateSymbolByHostname(OVwMapInfo *map,
       OVwSubmapId submapId, char *hostname,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
       OVwSymbolPosition *symbolPosition, unsigned int flags);

OVwSymbolId OVwCreateComponentSymbol(OVwMapInfo *map,
       OVwObjectId parentId, OVwObjectId objectId,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
```

```
       OVwSymbolPosition *symbolPosition, unsigned int flags);

OVwSymbolId OVwCreateComponentSymbolByName(OVwMapInfo *map,
       OVwObjectId parentId, OVwFieldBinding *name,
       OVwSymbolType symbolType, char *label,
       OVwStatusType status, int statusSource,
       OVwSymbolPosition *symbolPosition, unsigned int flags);

OVwSymbolId OVwCreateConnSymbol(OVwMapInfo *map,
       OVwObjectId objectId,
       OVwSymbolId endpoint1, OVwSymbolId endpoint2,
       OVwSymbolType symbolType, char *label, OVwStatusType status,
       int statusSource, unsigned int flags);

OVwSymbolId OVwCreateConnSymbolByName(OVwMapInfo *map,
       OVwFieldBinding *name,
       OVwSymbolId endpoint1, OVwSymbolId endpoint2,
       OVwSymbolType symbolType, char *label, OVwStatusType status,
       int statusSource, unsigned int flags);

int OVwDeleteSymbol(OVwMapInfo *map, OVwSymbolId symbolId);

int OVwDeleteSymbols(OVwMapInfo *map, OVwSymbolIdList *symbolIdList);
```

Description

OVwCreateSymbol and its related functions create and delete symbols on a submap of the open map. OVwCreateSymbols is the base function upon which all the other functions are built. The various convenience routines, such as OVwCreateSymbolByHostname, are provided to simplify the calling sequence and arguments in different situations.

There are two varieties of symbols: icon symbols (ovwIconSymbol) and connection symbols (ovwConnSymbol). The following routines can be used to create icon symbols:

- OVwCreateSymbols
- OVwCreateSymbol
- OVwCreateSymbolByName
- OVwCreateSymbolBySelectionName
- OVwCreateSymbolByHostname
- OVwCreateComponentSymbol
- OVwCreateComponentSymbolByName

OVwCreateConnSymbol and OVwCreateConnSymbolByName can be used to create connection symbols.

OVwCreateSymbol creates a symbol representing the object identified by objectId and adds it to the submap identified by submapId on the open map.

An application can create a symbol on any submap with the policy, ovwSharedSubmap. Only the application that created the submap can create a symbol on a submap with the policy, ovwExclusiveSubmap (see “OVwCreateSubmap(3)” on page 619.)

A symbol created on a submap by an application is placed on the application plane of the submap. The list of applications interested in a particular symbol is initialized with the application creating the symbol (the apps field of the OVwSymbolInfo structure).

There is a special root submap that is available to all applications for creating symbols that represent the parent object of the top level submap of a significant hierarchy of submaps. The intention of this submap is to provide a common place for multiple applications to anchor their submap hierarchies. You should create symbols on the root submap sparingly; only symbols representing very high-level compound objects should be added to the root submap. The submap ID for the root submap is available as the root_submap_id field of the OVwMapInfo structure for the open map. See "OVwMapOpenCB(3)" on page 761 for more information.

The optional symbolPosition parameter enables the specification of symbol-placement information. This parameter will normally be NULL. If symbolPosition is NULL, the symbol will be placed in the submap according to the automatic layout algorithm specified for the submap. If there is need for greater control over symbol placement in a submap, the symbolPosition parameter can be used. See "OVwSetSymbolPosition(3)" on page 815 for details on symbol placement. The effect of setting the position of a symbol when creating it is the same as setting the position of a symbol using OVwSetSymbolPosition.

OVwCreateSymbols creates multiple symbols with a single call. This is more efficient than making individual calls to create each symbol. OVwCreateSymbols creates both icon symbols and connection symbols. Icon symbols and the connection symbol that connects them can even be created in the same call, provided that the connection symbol appears later in the symbol list and uses the index fields of the OVwSymbolCreateInfo structure for referring to a symbol created earlier within the same call.

If the operation fails for any of the elements in the list, OVwCreateSymbols returns an error code Even if an error occurs, the operation will still be performed for all those elements on which it can. Upon return, the error field of the OVwSymbolCreateInfo structure will indicate which list elements failed. Also upon return, the symbol_id field of the OVwSymbolCreateInfo structure will contain the symbol ID of all those symbols that were successfully created.

OVwCreateSymbolByName creates a symbol representing the object having the name field value indicated by name and adds it to the submap of the open map identified by submapId. If an object with the specified name does not exist, it will be automatically created using OVwDbCreateObject. If name is NULL, an object will be automatically created with a system-generated name.

OVwCreateSymbolBySelectionName creates a symbol representing the object identified by the specified selection name. If an object with the specified selection name does not exist, it will be automatically created using OVwDbCreateObjectBySelectionName.

OVwCreateSymbolByHostname creates a symbol representing the object identified by the specified IP host name. If an object with the specified host name does not exist, it will be automatically created using OVwDbCreateObjectByHostname.

OVwCreateComponentSymbol creates a symbol representing the object identified by objectId on the child submap of a compound object identified by parentId. If the child submap of the object parentId does not exist, it is automatically created using OVwCreateSubmap with the submap policy, ovwSharedSubmap.

OVwCreateComponentSymbolByName creates a symbol representing the object, which is identified by name on the child submap of a compound object, which is identified by parentId. If the child submap of the object parentId does not exist, it is automatically created using OVwCreateSubmap with the submap policy, ovwSharedSubmap. If an object with the specified name does not exist, it will be automatically created using OVwDbCreateObject.

OVwCreateConnSymbol creates a connection symbol, representing an object identified by objectId between two icon symbols that are identified by endpoint1 and endpoint2 on the submap that is identified by submapId. If the layout of the submap on which the symbol is being created (ovwBusLayout, or

ovwRingLayout) has a backbone, one of the end points can be specified as ovwSubmapBackbone to indicate that the icon symbol should be connected to the ring or bus cable.

When the first connection is created between two symbols, a simple connection is created. Creating any additional connections between the two symbols results in the automatic creation of a metaconnection submap that contains the multiple simple connections between the symbols. This special submap, which is the child submap of a metaconnection object represented by a metaconnection symbol, has the submap policy, ovwMetaConnSubmap.

If the two end points have more than one connection between them, the symbol ID returned by OVwCreateConnSymbol identifies the connection symbol created in the metaconnection submap.

Because the metaconnection submap is intended to represent only connections between the two symbols in the parent submap, connections cannot be created directly in the metaconnection submap. A connection can only be added to the submap indirectly by creating a connection between the two symbols whose connections it represents. This prevents the recursion of metaconnection submaps. An application can create child submaps for objects represented by connection symbols in the metaconnection submap. See “OVwCreateSubmap(3)” on page 619.

OVwCreateConnSymbolByName creates a connection symbol representing an object identified by name between two icon symbols that are identified by endpoint1 and endpoint2 on a submap of the open map. If an object with the specified name does not exist, it will be automatically created using OVwDbCreateObject.

Symbol creation routines can result in the generation of the following events:

ovwConfirmCreateSymbols (OVwConfirmCreateSymbolsCB)
ovwConfirmCreateObjects (OVwConfirmCreateObjectsCB)
ovwConfirmCreateSubmaps (OVwConfirmCreateSubmapsCB)

OVwDeleteSymbol deletes the symbol identified by symbolId from the open map. Both icon and connection symbols are deleted using this routine. A symbol cannot be deleted from the application plane of an exclusive submap created by another application.

Deletion is a recursive operation. If the last symbol of an object is deleted, the object represented by the symbol is deleted from the map. If the deleted object has a child submap, the child submap and all its symbols are deleted, and so on. The following list provides the events that can be used to determine which symbols were deleted and their related callbacks:

ovwConfirmDeleteSymbols
OVwConfirmDeleteSymbolsCB

ovwConfirmDeleteSubmaps
OVwConfirmDeleteSubmapsCB

ovwConfirmDeleteObjects OVwConfirmDeleteObjectsCB

When the last symbol of an object is deleted from the open map, the object is automatically deleted from the open map. When an object has been deleted from the last map on which it appears, an ovwConfirmDeleteObjects event is generated with the op_scope field of the OVwObjectInfo structure set to ovwAllMapsScope. This notification that an object no longer exists on any maps is a signal to applications to call OVwDbUnsetFieldValue, for every field that the application maintains for the object, and to call OVwDbDeleteObject. If no other applications have fields set for the object, the object will be deleted from the OVW object database.

OVwDeleteSymbols deletes multiple symbols from the open map. This is more efficient than making individual calls to delete each symbol. OVwDeleteSymbols returns a single error if the call fails for any of the

elements in the list. Even if an error occurs, the operation will be performed for those elements on which it can. Individual error information for each element is not returned. If individual error information is needed, use OVwDeleteSymbol. A confirmation of which symbols were deleted is available through the ovwConfirmDeleteSymbols event.

Parameters

endpoint1 Specifies the symbol ID of an icon symbol that is a connection end point.

endpoint2 Specifies the symbol ID of an icon symbol that is a connection end point. The special value ovwSubmapBackbone can be used if the layout of the submap, ovwBusLayout or ovwRingLayout, includes a backbone.

flags Specifies symbol creation flags. This is the logical OR of the following flags defined in the <OV/ovw.h> header file:

ovwNoSymbolFlags This value can be specified if no flags are needed.

ovwMergeDefaultCapabilities
The default capability field values for the symbol type symbolType (as defined in a symbol type registration file) will be set on the symbol's object for those fields that do not already have values set.

ovwDoNotDisplayLabel The symbol label will not be displayed. This flag should normally be set for connection symbols.

ovwDeleteDescendants When this symbol is deleted, all other symbols that represent this same object on submaps descending from this symbol's submap will also be deleted. This flag is useful for applications that build a submap hierarchy with symbols representing the same object appearing on several submaps. This flag facilitates deletion of the object, since all symbols representing an object must be deleted before the object can be deleted.

hostname Specifies a pointer to the official IP host name of the object represented by the symbol.

label Specifies a pointer to the symbol label. This parameter will normally be NULL when a connection symbol is created.

map Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

name Specifies a pointer to the name of the object that the symbol represents.

objectId Specifies the object ID of the object that the symbol represents.

parentId Specifies the object ID of the parent object of a submap on which to create a symbol.

selectionName Specifies a pointer to the selection name of the object that the symbol represents.

status Specifies the initial status of the symbol. The permitted values are defined in the <OV/ovw_types.h> header file:

ovwUnknownStatus	The status is unknown.
ovwNormalStatus	The status is up or normal.
ovwMarginalStatus	The status is marginal (some problem exists).
ovwCriticalStatus	The status is down or critical.
ovwUnmanagedStatus	The object should be created on the map as unmanaged. This value will be ignored unless this is the first symbol for the object on the map. Unmanaged objects are not monitored and do not show status.

The status argument is used only if statusSource is ovwSymbolStatusSource, except in the following situations:

- If status is ovwUnmanagedStatus and this is the first symbol for the object, the object becomes unmanaged.
- If statusSource is ovwObjectStatusSource and this is the first symbol for the object, status is used to initialize the object status.

statusSource Specifies the status source for the symbol. The permitted values are defined in the <OV/ovw.h> header file:

ovwObjectStatusSource	The symbol gets its status from the status of the object.
ovwCompoundStatusSource	The symbol gets its status through propagation from the child submap of the object.
ovwSymbolStatusSource	The symbol has its status set explicitly.

submapId Specifies the submap ID of the submap on which to create the symbol.

symbolId Specifies the symbol ID of the symbol to delete.

symbolIdList Specifies a pointer to the list of symbol IDs for the symbols to delete.

symbolList Specifies a pointer to the list of symbols to create.

symbolPosition Specifies a pointer to optional symbol position information structure. This parameter will normally be NULL, which uses the automatic layout algorithm for the submap. See “OVwSetSymbolPosition(3)” on page 815 for more information on the use of this parameter.

symbolType Specifies the symbol type to use for displaying the symbol. Symbol type values are defined in the symbol type registration files. Some predefined symbol types are also listed in the header file <OV/sym_types.h>. For connection symbols, a value of NULL can be used to indicate the default connection symbol type. A NULL value is not allowed for icon symbols. Symbol types of the Connection symbol class can be used for connection symbols.

Return Values

If successful, the following functions return a valid symbol ID:

- OVwCreateSymbol
- OVwCreateSymbolByName
- OVwCreateSymbolBySelectionName
- OVwCreateSymbolByHostname
- OVwCreateComponentSymbol
- OVwCreateComponentSymbolByName
- OVwCreateConnSymbol
- OVwCreateConnSymbolByName

If unsuccessful, they return ovwNullSymbolId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing symbol IDs.

If successful, OVwCreateSymbols, OVwDeleteSymbol, and OVwDeleteSymbols return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwCreateSymbol and its related functions set the error code that OVwError returns.

The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is opened with read-only permission.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_PERMISSION_DENIED]	The submap on which the call is being made is an exclusive submap that was created by another application.

The following additional errors may result from create symbol routines:

[OVw_FIELD_NOT_FOUND]	The field, specified by field_id in the OVwFieldBinding, does not exist.
[OVw_FIELD_NOT_NAME]	The field, specified by field_id in the OVwFieldBinding, does not represent a name field (ovwNameField).
[OVw_FIELD_TYPE_MISMATCH]	The field type, of the field identified by field_id in the OVwFieldBinding structure, has a type different from the OVwFieldValue structure.
[OVw_FIELD_VALUE_NULL]	The field_val field of the OVwFieldBinding structure is NULL.
[OVw_OBJECT_NOT_FOUND]	The object, specified by objectId, does not exist.
[OVw_OBJECT_NULL_NAME]	The selectionName or hostname argument is NULL.

[OVw_PARENT_OBJECT_NOT_FOUND]	The parent object specified by parentId does not exist.
[OVw_SUBMAP_NOT_FOUND]	The submap, specified by submapId, does not exist on the open map.
[OVw_SYMBOL_INVALID_FLAGS]	The argument flags, or the flags field of the OVwSymbolCreateInfo structure for OVwCreateSymbols, has a value that is not valid.
[OVw_SYMBOL_INVALID_OBJECT_NAME_STYLE]	The object_name_style field of the OVwSymbolCreateInfo structure (for OVwCreateSymbols) has a value that is not valid.
[OVw_SYMBOL_INVALID_STATUS]	The argument status, or the status field of the OVwSymbolCreateInfo structure for OVwCreateSymbols, has a value that is not valid .
[OVw_SYMBOL_INVALID_STATUS_SOURCE]	The argument statusSource, or the status_source field of the OVwSymbolCreateInfo structure for OVwCreateSymbols, has a value that is not valid.
[OVw_SYMBOL_INVALID_SUBMAP_NAME_STYLE]	The submap_name_style field of the OVwSymbolCreateInfo structure, for OVwCreateSymbols, has a value that is not valid.
[OVw_SYMBOL_INVALID_VARIETY]	The symbol_variety field of the OVwSymbolCreateInfo structure, for OVwCreateSymbols, has a value that is not valid.
[OVw_SYMBOL_TYPE_NOT_FOUND]	The argument symbolType does not specify a registered symbol type.
[OVw_SYMBOL_TYPE_WRONG_VARIETY]	The argument symbolType or the symbol_type field of the OVwSymbolCreateInfo structure, for OVwCreateSymbols, has a symbol type variety (icon or connection) that does not match the variety of the symbol specified by symbolId. The variety of a symbol type is determined by the variety of its symbol class as defined in the symbol type registration file.

The following errors can result from calls to create icon symbols:

[OVw_ICON_SYMBOL_BAD_COORDS]	The x or y coordinate specified in the position argument has a value that is less than zero or greater than the width or height of the grid coordinate system.
[OVw_ICON_SYMBOL_BAD_GRID]	The width or height specified in the position argument for setting the scale for the x and y coordinates has a value less than or equal to zero.
[OVw_ICON_SYMBOL_PRED_NOT_FOUND]	The symbol specified in the position argument as the predecessor of the symbol symbolId does not exist on the same submap as the symbol symbolId.
[OVw_SUBMAP_INVALID_SYMBOL_PLACEMENT]	The placement field of the position argument has a value that is not valid for the layout of the submap on which the symbol symbolId exists.

The following errors can result from calls to create connection symbols:

[OVw_CONN_SYMBOL_BOTH_ENDS_NULL] Both connection end points have the value ovwNullSymbolId.

[OVw_CONN_SYMBOL_BOTH_ENDS_SAME] Both connection end points are the same.

[OVw_CONN_SYMBOL_END_NOT_FOUND] One of the connection end-point symbols does not exist on the open map.

[OVw_CONN_SYMBOL_END_WRONG_VARIETY]
The variety of a connection end-point symbol is valid. Only icon symbols are allowed as connection end points.

[OVw_CONN_SYMBOL_ENDS_DIFFERENT_SUBMAPS]
The two connection end points are on different submaps. Both connection end points must be on the same submap.

[OVw_CONN_SYMBOL_INVALID_END_NAME_STYLE]
One of the end-point name style fields of the OVwSymbolCreateInfo structure, for OVwCreateSymbols, has a value that is not valid.

[OVw_CONN_SYMBOL_META_CONN_SUBMAP]
An attempt was made to create a connection directly on a metaconnection submap.

[OVw_CONN_SYMBOL_NO_SUBMAP_BACKBONE]
A value of ovwSubmapBackbone is specified as a connection end point for a submap that does not have a backbone.

The OVwDeleteSymbol and OVwDeleteSymbols routines return the following additional error:

[OVw_SYMBOL_NOT_FOUND] The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwCreateSymbol and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwCreateSymbol function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See “OVwConfirmCreateSymbolsCB(3)” on page 586.
- See “OVwConfirmCreateObjectsCB(3)” on page 582.
- See “OVwConfirmCreateSubmapsCB(3)” on page 584.
- See “OVwConfirmDeleteSubmapsCB(3)” on page 590.
- See “OVwConfirmDeleteObjectsCB(3)” on page 588.
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbDeleteObject(3)” on page 641.
- See “OVwDbUnsetFieldValue(3)” on page 681.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See “OVwMapOpenCB(3)” on page 761.
- See “OVwSetSymbolPosition(3)” on page 815.
- See “OVwVerifyAdd(3)” on page 833.
- See “OVwVerifyDeleteSymbol(3)” on page 846.
- See “OVwApiIntro(5)” on page 560.

OVwDbAppendEnumConstants(3)

Purpose

Appends constants to an existing enumeration

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbAppendEnumConstants(OVwFieldId fieldId,
       OVwEnumConstants *enumConstants);
```

Description

OVwDbAppendEnumConstants provides the ability to add enumerated constants to an already defined set. The new constants listed in enumConstants are appended, in the order received, to the end of the enumeration currently set for the field. If no enumeration has been set for the field, this function performs like OVwDbSetEnumConstants.

See “OVwDbSetEnumConstants(3)” on page 674 for more information on setting enumerated constants.

Parameters

enumConstants	Specifies a pointer to an OVwEnumConstants structure.
fieldId	Uniquely identifies an object attribute field. This ID must represent a field in the OVW object database that was created with a data type of ovwEnumField.

Return Values

If successful, OVwDbAppendEnumConstants returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwDbAppendEnumConstants sets the error code that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NOT_FOUND]	The fieldId does not identify a field in the OVW object database.
[OVw_FIELD_TYPE_MISMATCH]	The type of the field identified by fieldId is not ovwEnumField.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbAppendEnumConstants supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbAppendEnumConstants, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See OVwDbGetFieldEnumByValue in “OVwDbGetFieldValue(3)” on page 650.
- See OVwDbGetFieldEnumByName in “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetEnumConstants(3)” on page 674.
- See OVwDbSetFieldEnumByName in “OVwDbSetFieldValue(3)” on page 676.
- See OVwDbSetFieldEnumByValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbCreateField(3)

Purpose

Creates a field in the object database

Related Functions

OVwDbDeleteField

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldId OVwDbCreateField(char *fieldName, int fieldType,
       unsigned int fieldFlags);

int OVwDbDeleteField(OVwFieldId fieldid);
```

Description

OVwDbCreateField creates a new field in the OVW object database. The OVwFieldId returned from this call is a unique handle used to identify the field.

OVwDbDeleteField removes the field from the OVW object database. Removing a field from the database has potential global impact. The field definition is removed and is no longer valid for setting or retrieving data. In addition, any value for this field associated with objects is removed from the object database. Some predefined fields may not be removed from the database.

Parameters

fieldFlags

Specifies a bitmap representing what field flags are to be set for the newly created field. The following permitted values are defined in <OV/ovw_obj.h>:

ovwListField	Enables a field to take on a list of values rather than a single value. This flag may be set for the ovwStringField and the ovwIntegerField data types.
ovwNameField	Enables the field value to be set to become a unique handle for the object. This flag can only be set with the ovwStringField data type. A name-field value must be unique for all values set using this field.
ovwCapabilityField	Specifies that the field is considered by the NetView for AIX program to be an object capability. This flag can only be set for the ovwBooleanField and the ovwIntegerField data types.
ovwLocateField	Indicates to the NetView for AIX program that this field may be used to locate objects for the user interface. By setting this flag, the field name will appear in the Locate by Attribute dialog box. This flag can be used with fields of any data type.
ovwGeneralField	Provides the ability to have nonapplication-specific fields that do not appear in any application-specific dialog box appear in the special Attributes for Object dialog box associated with every object. This flag can be used with fields of any data type.

fieldId
Specifies the field ID of the field to be deleted.

fieldName
Specifies a pointer to a NULL-terminated character string representing a textual name that will uniquely identify the newly created field. The fieldName parameter must be unique over all field names in the OVW object database. A one-to-one correspondence exists between field ID to the provided field name.

fieldType
Specifies the data type associated with the new field. Valid field types are ovwIntField, ovwBooleanField, ovwEnumField, and ovwStringField. The value set for this parameter dictates the type of data that may be set for an object for this field.

Return Values

If successful, OVwDbCreateField returns a value field ID. If unsuccessful, it returns ovwNullFieldId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing field IDs.

If successful, OVwDbDeleteField returns 1. If unsuccessful, it returns 0 (zero) or –1 (negative one). If OVwDbDeleteField is successful, fieldName no longer represents a valid field definition.

Error Codes

OVwDbCreateField and OVwDbDeleteField set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED] An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

OVwDbCreateField returns the following errors:

[OVw_FIELD_EXISTS] A field with the provided name exists in the database.

[OVw_FIELD_INVALID_TYPE] The value defining the type to be set for the new field is not one of the valid types.

[OVw_FIELD_INVALID_FLAG] The value defining the flags to be set for the new field does not represent valid flags.

OVwDbDeleteField returns the following errors:

[OVw_FIELD_NOT_FOUND] The provided field ID does not represent any field in the database.

Implementation Specifics

OVwDbCreateField supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbCreateField or OVwDbDeleteField, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbFieldNameToFieldId(3)” on page 643.
- See “OVwDbGetFieldInfo(3)” on page 648.
- See “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbListFields(3)” on page 664.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See OVwIsIdEqual in “OVwIsIdNull(3)” on page 743.
- See “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbCreateObject(3)

Purpose

Creates an object in the OVW object database

Related Functions

OVwDbCreateObjectBySelectionName
OVwDbCreateObjectByHostname

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectId OVwDbCreateObject(OVwFieldBinding *name);

OVwObjectId OVwDbCreateObjectBySelectionName(char *selectionName);

OVwObjectId OVwDbCreateObjectByHostname(char *hostname);

OVwRetCodeList *OVwDbCreateObjectWithFields(OVwObjectsFieldBindList *objectsFieldBindList);

void OVwDbFreeRetCodeList(OVwRetCodeList *retCodeList);
```

Description

OVwDbCreateObject creates a new object in the OVW Object database and assigns a value to a single field for the object. If the field specified in the OVwFieldBinding is not the selection name field for the object, a selection name is automatically created and set for the object. An error results if the selection name is provided but is not unique. If NULL is provided as input, OVwDbCreateObject creates an object, and sets the selection name to a system-generated name.

OVwDbCreateObjectBySelectionName creates a new object in the OVW Object database. An error occurs if an object already exists with the selection name. If NULL is provided as input, OVwDbCreateObject creates an object and sets the selection name to a system-generated name.

OVwDbCreateObjectByHostname creates a new object in the OVW Object database. The selection name of the object is set automatically. An error results if the provided host name is NULL or not unique for all objects in the database.

OVwDbCreateObjectWithFields creates objects and sets fields on those objects as specified by the objectsFieldBindList parameter. This routine creates multiple objects and sets fields for those objects, the same as if the OVwDbCreateObject and OVwDbSetFieldValues routines were used several times. Using OVwDbCreateObjectsWithFields to create and set fields for mutliple objects is much faster than creating each object and setting its fields separately with OVwDbCreateObject and OVwDbSetFieldValues.

The first field of the OVwFieldBindList of the OVwObjectsFieldBindList parameter is a name field. This field is used to create the object, just as if OVwDbCreateObject had been used. The rest of the fields in OVwFieldBindList are then set as if OVwDbSetFieldValues had been used.

OVwDbFreeRetCodeList frees memory allocated for an OVwRetCodeList structure. A pointer to an OVwRetCodeList structure is returned by a successful call to OVwDbCreateObjectsWithFields. This

pointer should be used as the retCodeList parameter for OVwDbFreeRetCodeList to free the structure when it is no longer needed.

Parameters

hostname Specifies a pointer to the host name, which is to be set for the newly created object.

name Specifies a pointer to an OVwFieldBinding structure, which contains the field to be set for the new object.

objectsFieldBindList
Specifies a list of OVwFieldBindLists. The first OVwFieldBinding of each OVwFieldBindList is used to create an object, therefore the first field should be a name field. The remaining OVwFieldBindings of each OVwFieldBindList describe field values that are set on the objects created with the name fields described by the first OVwFieldBinding in the OVwFieldBindLists.

retCodeList Specifies a pointer to an OVwRetCodeList structure. This structure is returned by a successful call to OVwDbCreateObjectsWithFields.

selectionName Specifies a pointer to the selection name of the object. The selectionName must be unique for all objects.

Return Values

If successful, OVwDbCreateObject, OVwDbCreateObjectBySelectionName, and OVwDbCreateObjectByHostName return the valid object ID of the newly created object. If unsuccessful, they return ovwNullObjectId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing submap IDs.

If successful, OVwDbCreateObjectsWithFields returns a pointer to an OVwRetCodeList structure. This structure contains a list of OVwRetCodeInfo structures which contain return codes for each object creation attempted by OVwDbCreateObjectsWithFields. Free the pointer to the OVwRetCodeList by calling OVwFreeRetCodeInfo. If unsuccessful, OVwDbCreateObjectsWithFields returns NULL.

Error Codes

OVwDbCreate, OVwDbCreateObjectBySelectionName, and OVwDbCreateObjectByHostname set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST] The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED] An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_FIELD_NAME_NOT_UNIQUE] The name provided is not unique over all values set for this field. This error will only be seen for fields that have the ovwNameField flag set.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

OVwDbCreateObject returns the following errors:

[OVw_FIELD_NOT_FOUND] The name specifies a field that does not exist in the OVW object database.

[OVw_FIELD_TYPE_MISMATCH] The field type of the field does not match the field type specified in the OVwFieldValue structure.

OVwDbCreateObjectByHostname returns the following error:

[OVw_FIELD_VALUE_NULL] The name provided to the function is NULL.

Implementation Specifics

OVwDbCreateObject supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbCreateObject or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbGetUniqObjectName(3)” on page 658.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbDeleteObject(3)

Purpose

Deletes an object from the OVW object database

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbDeleteObject(OVwObjectId objId);
```

Description

OVwDbDeleteObject deletes an object from the OVW object database, unless another application is using the object. Before calling OVwDbDeleteObject, an application should call OVwDbUnsetFieldValue for all the fields for which it sets values. No application is considered to be using the object if the only fields set for the object are the Selection Name, capability fields, or general fields (fields with the ovwCapabilityField or the ovwGeneralField flag set). If these are the only remaining fields set for the object, the object is deleted. Otherwise, the object is not deleted, because other applications have fields set for the object. Thus, if multiple applications are sharing an object, the object will only be deleted when the last application unsets its fields from the object and calls delete.

Also, before deleting an object, all symbols representing that object must be deleted from all submaps of all maps.

Parameters

objId Specifies the object ID of the object to delete.

Return Values

If successful, OVwDbDeleteObject returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwDbDeleteObject sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
: The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED] An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_UNABLE_TO_DELETE_OBJECT]
: The object could not be deleted, because fields without the ovwCapabilityField or the ovwGeneralField field flags remain set for the object indicating another application still has interest in the object. An application receiving this error should continue processing. The last application to delete the object will be successful.

Examples

The following example illustrates how an application uses OVwDbDeleteObject once the application is finished with the object:

```
for (each field the application is responsible for) {
    /*
     * Unset all the field values the application is responsible for
     */
    OVwDbUnsetFieldValue(object_id,field_id);
}

/*
 * Once all the field values have been unset, attempt to delete
 * the object.
 */
OVwDbDeleteObject(object_id);
```

Implementation Specifics

OVwDbDeleteObject supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbDeleteObject, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbUnsetFieldValue(3)” on page 681.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbFieldNameToFieldId(3)

Purpose

Converts a field name to a field ID

Related Functions

OVwDbFieldIdToFieldName

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldId OVwDbFieldNameToFieldId(char *fieldName);

char *OVwDbFieldIdToFieldName(OVwFieldId fieldId);
```

Description

Each field in the OVW object database can be uniquely identified by field ID or a character string representing the field name. There is a one-to-one relationship between a field name and a field ID. These routines enable the conversion between field name and field ID. See “OVwDbCreateField(3)” on page 635 for more information on field names and field IDs.

OVwDbFieldNameToFieldId returns the field ID of the field that has the field name fieldName.

OVwDbFieldIdToFieldName returns the name of the field that has field ID fieldId.

Parameters

fieldId Specifies a field ID.

fieldName Specifies a pointer to a NULL-terminated string uniquely identifying a field in the OVW object database.

Return Values

If successful, OVwDbFieldNameToFieldId returns the field ID that uniquely identifies the field with the name fieldName. If unsuccessful, it returns ovwNullFieldId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing field IDs.

If successful, OVwDbFieldIdToFieldName returns a string specifying the name of the field that is uniquely identified by fieldId. If unsuccessful, it returns NULL. Because the return value for OVwDbFieldIdToFieldName is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbFieldNameToFieldId and OVwDbFieldIdToFieldName set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NOT_FOUND]	Depending on the function being called, either the provided field name or the provided field ID does not represent a field in the OVW object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbFieldNameToFieldId and OVwDbFieldIdToFieldName support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbFieldNameToFieldId or OVwDbFieldIdToFieldName, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbGetFieldInfo(3)” on page 648.
- See “OVwDbInit(3)” on page 662.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetEnumConstants(3)

Purpose

Accesses enumerated type values

Related Functions

OVwDbFreeEnumConstants

OVwDbGetEnumValue

OVwDbGetEnumName

Syntax

```
#include <OV/ovw_obj.h>

OVwEnumConstants *OVwDbGetEnumConstants(OVwFieldId fieldId);

void OVwDbFreeEnumConstants(OVwEnumConstants *enumConstants);

int OVwDbGetEnumValue(OVwFieldId fieldId, char *name);

char *OVwDbGetEnumName(OVwFieldId fieldId, int value);
```

Description

OVwDbGetEnumConstants returns a list of constants in the enumerated data type.

OVwDbFreeEnumConstants frees the memory allocated for an OVwEnumConstants structure.

OVwDbGetEnumValue translates an enumerated constant into an index value.

OVwDbGetEnumName translates an index into an enumerated constant.

Parameters

enumConstants
Specifies a pointer to a structure of type OVwEnumConstants.

fieldId
Specifies the field ID of the enumerated type. This ID must represent a field in the NetView for AIX object database that was created with the data type ovwEnumField.

name
Specifies a pointer to a text value defined as an enumeration constant for the field.

value
Specifies the index number of an enumeration constant for the field.

Return Values

If successful, OVwDbGetEnumConstants returns a pointer to an OVwEnumConstants structure. The count field in the OVwEnumConstants structure represents the number of entries that have been defined in the enumerated type. A count of 0 (zero) indicates that no values have been set for that enumerated type. If unsuccessful, OVwDbGetEnumConstants returns NULL.

If successful, OVwDbGetEnumValue returns the index value for name in the enumerated data type. If unsuccessful, it returns –1 (negative one).

If successful, OVwDbGetEnumName returns the text value (name) that has the index value for the field. If unsuccessful, it returns NULL. Because the return value for OVwDbGetEnumName is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbGetEnumConstants and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the NetView for AIX object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NOT_FOUND]	The argument fieldId does not identify a field in the NetView for AIX object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

OVwDbGetEnumValue and OVwDbGetEnumName return the following error:

[OVw_FIELD_TYPE_MISMATCH]	The type of the field identified by fieldID is not ovwEnumField.

OVwDbGetEnumValue returns the following error:

[OVw_INDEX_OUT_OF_RANGE]	The argument name could not be found in the range of the enumerated type identified by fieldId.

OVwDbGetEnumName returns the following error:

[OVw_NAME_NOT_FOUND]	The argument value could not be found in the range of the enumerated type identified by fieldId.

Libraries

When compiling a program that uses OVwDbGetEnumConstants or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See OVwDbGetFieldEnumByValue in “OVwDbGetFieldValue(3)” on page 650.
- See OVwDbGetFieldEnumByName in “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetEnumConstants(3)” on page 674.
- See OVwDbSetFieldEnumByName in “OVwDbSetFieldValue(3)” on page 676.
- See OVwDbSetFieldEnumByValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetFieldInfo(3)

Purpose

Returns information about a database field

Related Functions

OVwDbFreeFieldInfo

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldInfo *OVwDbGetFieldInfo(OVwFieldId fieldId);

void OVwDbFreeFieldInfo(OVwFieldInfo *fieldBuff);
```

Description

OVwDbGetFieldInfo returns information about the field identified by fieldId. Returned information includes the field name, field data type, and field flag information. For more information on how this information is set for a field, see “OVwDbCreateField(3)” on page 635.

OVwDbFreeFieldInfo frees the memory allocated for an OVwFieldInfo structure.

Parameters

fieldBuff	Specifies a pointer to the OVwFieldInfo structure that is to be freed
fieldId	Specifies the ID of the field

Return Values

If successful, OVwDbGetFieldInfo returns a pointer to the OVwFieldInfo structure that contains the relevant field information. If unsuccessful, OVwDbGetFieldInfo returns NULL.

Error Codes

OVwDbGetFieldInfo sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The OVw API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NOT_FOUND]	The fieldId does not represent a field in the OVW object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbGetFieldInfo supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbGetFieldInfo or OVwDbFreeFieldInfo, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See OVwDbFieldIdToFieldName in “OVwDbFieldNameToFieldId(3)” on page 643.
- See “OVwDbFieldNameToFieldId(3)” on page 643.
- See “OVwDbInit(3)” on page 662.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetFieldValue(3)

Purpose

Gets field values for an object

Related Functions

OVwDbGetFieldEnumByValue
OVwDbGetFieldEnumByName
OVwDbGetFieldBooleanValue
OVwDbGetFieldIntegerValue
OVwDbGetFieldStringValue
OVwDbFreeFieldValue

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldValue *OVwDbGetFieldValue(OVwObjectId objectId,
       OVwFieldId fieldId);

int OVwDbGetFieldEnumByValue(OVwObjectId objectId,
       OVwFieldId fieldId);

char *OVwDbGetFieldEnumByName(OVwObjectId objectId,
       OVwFieldId fieldId);

OVwBoolean OVwDbGetFieldBooleanValue(OVwObjectId objectId,
       OVwFieldId fieldId);

int32 OVwDbGetFieldIntegerValue(OVwObjectId objectId,
       OVwFieldId fieldId);

char *OVwDbGetFieldStringValue(OVwObjectId objectId,
       OVwFieldId fieldId);

void OVwDbFreeFieldValue(OVwFieldValue *fieldValue);
```

Description

OVwDbGetFieldValue returns a pointer to an OVwFieldValue structure for the field identified by fieldId for the object identified by objectId.

For a given object OVwDbGetFieldEnumByValue returns the value set for a field of type ovwEnumField. The value returned is an index into the enumerated type, which is associated with the field identified by fieldId for an object identified by objectId.

For a given object OVwDbGetFieldEnumByName returns the value set for a field of type ovwEnumField. The value returned represents the actual value name stored in the enumerated type, which is associated with the field defined by fieldId for the object identified by objectId.

OVwDbGetFieldBooleanValue returns the boolean value set for a field of type ovwBooleanField identified by fieldId for the object identified by objectId.

OVwDbGetFieldIntegerValue returns the integer value set for a field of type ovwIntField identified by fieldId for the object identified by objectId. OVwDbGetFieldIntegerValue cannot be used to return values assigned to fields with the ovwListField flag set.

OVwDbGetFieldStringValue returns the string value set for a field of type ovwStringField identified by fieldId for the object identified by objectId. Because the return value for OVwDbGetFieldStringValue is dynamically allocated, you must free the string when it is no longer needed. OVwDbGetFieldStringValue cannot be used to return values assigned to fields with the ovwListField flag set.

OVwDbFreeFieldValue frees the memory allocated for an OVwFieldValue structure.

Parameters

fieldId	Specifies the ID of the field
fieldValue	Specifies a pointer to the OVwFieldValue structure to be freed
objectId	Specifies the ID of the object with the field value

Return Values

If successful, OVwDbGetFieldValue returns a pointer to an OVwFieldValue structure. If unsuccessful, OVwDbGetFieldValue returns NULL.

If successful, OVwDbGetFieldEnumByValue returns the enumerated index, which is set as a field value of the object identified by objectId. If unsuccessful, OVwDbGetFieldEnumByValue returns –1 (negative one).

If successful, OVwDbGetFieldEnumByName returns a pointer to the name which is set as a field value for the object identified by objectId. If unsuccessful, OVwDbGetFieldEnumByName returns NULL. Because the return value for OVwDbGetFieldEnumByName is dynamically allocated, you must free the string when it is no longer needed.

If successful, OVwDbGetFieldBooleanValue returns an OVwBoolean value. If unsuccessful, OVwDbGetFieldBooleanValue returns –1; in this case, OVwError is set to a value other than [OVw_SUCCESS].

If successful, OVwDbGetFieldIntegerValue returns the integer value that was assigned to the field. If unsuccessful, OVwDbGetFieldIntegerValue returns –1 (negative one). In this case, OVwError is set to a value other than [OVw_SUCCESS].

If successful, OVwDbGetFieldStringValue returns the string value. If unsuccessful, it returns NULL. Because the return value for OVwDbGetFieldStringValue is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbGetFieldValue and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_OBJECT_NOT_FOUND]	The object identified by objectId does not exist.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

The following routines:

- OVwDbGetFieldValue
- OVwDbGetFieldEnumByValue
- OVwDbGetFieldEnumByName
- OVwDbGetFieldBooleanValue
- OVwDbGetFieldIntegerValue
- OVwDbGetStringValue
- OVwDbGetValuesByObjects

return the following error:

[OVw_FIELD_NOT_FOUND]	The field identified by fieldId does not exist.

The following routines:

- OVwDbGetFieldEnumByValue
- OVwDbGetFieldEnumByName
- OVwDbGetFieldBooleanValue
- OVwDbGetFieldIntegerValue
- OVwDbGetStringValue

return the following errors:

[OVw_FIELD_TYPE_MISMATCH]	A field identified by fieldId does exist in the database, but has a type inconsistent with the call being used.
[OVw_FIELD_LIST_FLAG_SET]	An attempt was made to get the value of a list field.

The following routines:

- OVwDbGetFieldValue
- OVwDbGetFieldEnumByValue
- OVwDbGetFieldEnumByName
- OVwDbGetFieldBooleanValue
- OVwDbGetFieldIntegerValue
- OVwDbGetStringValue

return the following error:

[OVw_FIELD_VALUE_NULL]	The field does not have a value for this object.

Implementation Specifics

OVwDbGetFieldValue and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses an OVwDbGetFieldValue function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbGetFieldValues(3)” on page 654.
- See “OVwDbGetFieldValuesByObjects(3)” on page 656.
- See “OVwDbInit(3)” on page 662.
- See OVwDbSetFieldBooleanValue in “OVwDbSetFieldValue(3)” on page 676.
- See OVwDbSetFieldStringValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetFieldValues(3)

Purpose

Gets list of field values for an object

Related Functions

OVwDbGetCapabilityFieldValues
OVwDbGetNameFieldValues
OVwDbFreeFieldBindList

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldBindList *OVwDbGetFieldValues(OVwObjectId objectId);

OVwFieldBindList *OVwDbGetCapabilityFieldValues(OVwObjectId objectId);

OVwFieldBindList *OVwDbGetNameFieldValues(OVwObjectId objectId);

void OVwDbFreeFieldBindList(OVwFieldBindList *fieldList);
```

Description

OVwDbGetFieldValues returns a list of all the field values for the object identified by objectId. An object always has at least one field.

OVwDbGetCapabilityFieldValues returns a list of all field values for capability fields (fields with the ovwCapabilityField flag set) for the object identified by objectId.

OVwDbGetNameFieldValues returns a list of all field values for name fields (fields with the ovwNameField flag set) for the object identified by objectId. An object always has at least one name field.

OVwDbFreeFieldBindList frees the memory allocated for an OVwFieldBindList structure.

Parameters

fieldList	Specifies a pointer to an OVwFieldBindList structure.
objectId	Specifies the object ID of an object.

Return Values

If successful, OVwDbGetFieldValues returns a pointer to an OVwFieldBindList. If unsuccessful, it returns NULL.

If successful, OVwDbGetCapabilityFieldValues returns a pointer to an OVwFieldBindList structure. A value of 0 (zero) in the count field of the OVwFieldBindList structure indicates there are no capability fields set for the object. If unsuccessful, OVwDbGetCapabilityFields returns NULL.

If successful, OVwDbGetNameFieldValues returns a pointer to an OVwFieldBindList structure. If unsuccessful, it returns NULL.

Error Codes

OVwDbGetFieldValues and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_OBJECT_NOT_FOUND]	The object identified by objectId does not exist.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbGetFieldValues and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses an OVwDbGetFieldValues function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbGetFieldValuesByObjects(3)” on page 656.
- See “OVwDbInit(3)” on page 662.
- See OVwDbSetFieldBooleanValue in “OVwDbSetFieldValue(3)” on page 676.
- See OVwDbSetFieldStringValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetFieldValuesByObjects(3)

Purpose

Gets values for a field for multiple objects

Related Functions

OVwDbFreeObjectFieldList

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectFieldList *OVwDbGetFieldValuesByObjects(OVwObjectIdList *objectIdList,
       OVwFieldId fieldId);

void OVwDbFreeObjectFieldList(OVwObjectFieldList *objectFieldList);
```

Description

OVwDbGetFieldValuesByObjects is used to get the value that is set for a field for a list of objects. OVwObjectFieldList will point to a list of field values, one for each object represented in objectIdList. If for any given object ID the field has no value, the OVwFieldValue pointer for the object ID in the OVwObjectFieldList will be set to NULL.

OVwDbFreeObjectFieldList frees the memory allocated for an OVwObjectFieldList structure.

Parameters

fieldId	Specifies the ID of the field
objectFieldList	Specifies a pointer to an OVwObjectFieldList structure to be freed
objectIdList	Specifies a pointer to an OVwObjectIdList structure that contains a list of object IDs

Return Values

If successful, OVwDbGetFieldValuesByObjects returns a pointer to an OVwObjectFieldList structure. If unsuccessful, it returns NULL.

Error Codes

OVwDbGetFieldValuesByObjects and OVwDbFreeObjectFieldList set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_FIELD_NOT_FOUND]	The field identified by fieldId does not exist.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbGetFieldValuesByObjects supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbGetFieldValuesByObjects or OVwDbFreeObjectFieldList, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbInit(3)” on page 662.
- See OVwDbSetFieldBooleanValue in “OVwDbSetFieldValue(3)” on page 676.
- See OVwDbSetFieldStringValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbGetUniqObjectName(3)

Purpose

Gets a unique name for an object

Syntax

```
#include <OV/ovw_obj.h>

char *OVwDbGetUniqObjectName(OVwFieldId namefieldId,
       char *nameValue);
```

Description

The value set for a name field (a field with the ovwNameField flag set), must be unique within the OVW database. That is, each object must have a unique value for that field ID. OVwDbGetUniqObjectName returns a name value that is unique for all values set for the field identified by namefieldId.

The optional parameter nameValue can be used to determine whether a particular name is unique. among all values defined for the field specified by namefieldId. If the name value is found to be unique, it will be returned by the function unchanged. Otherwise, the name will be modified to ensure that it is unique. This parameter provides the ability to produce unique-name values seeded with a common name. If a NULL character string is provided through the nameValue parameter, a NetView for AIX internal name will be generated.

Parameters

namefieldId
Specifies the field ID for which a unique name is to be produced. The namefieldId parameter must identify a field that was created with the ovwNameField flag set.

nameValue
Specifies a pointer to a textual name value to be determined unique or not unique. The nameValue parameter can be NULL.

Return Values

If successful, OVwDbGetUniqObjectName returns a pointer to a character string representing a unique name. If unsuccessful, it returns NULL. Because the return value is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbGetUniqObjectName sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED] An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_FIELD_NOT_FOUND]	The field identified by namefieldId does not exist.
[OVw_FIELD_NOT_NAME]	The namefieldId parameter does not represent a field that has the ovwNameField flag set.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbGetUniqObjectName supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbGetUniqObjectName, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbNameToObjectId(3)” on page 670.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbHostnameToObjectId(3)

Purpose

Converts an IP host name to an object ID

Related Functions

OVwDbObjectIdToHostname

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectId OVwDbHostnameToObjectId(char *hostname);

char *OVwDbObjectIdToHostname(OVwObjectId objectId);
```

Description

The OVW object database defines the field IP Hostname. This field is created as a name field, ovwNameFieldSet, and is intended to provide database support for IP host names. These routines provide a convenient way to convert an IP host name to an object ID or to convert an object ID to an IP host name. Because IP Hostname is a name field, each host name set using this field uniquely identifies an object in the OVW database.

OVwDbHostnameToObjectId returns the object ID for the object whose IP host name is hostname.

OVwDbObjectIdToHostname returns the IP host name for the object identified by objectId.

Parameters

hostname	Specifies a pointer to the IP host name of an object
objectId	Specifies the object ID of an object

Return Values

If successful, OVwDbHostnameToObjectId returns the OVwObjectId that uniquely identifies the object that has hostname set for its IP Hostname field. If unsuccessful, it returns ovwNullObjectId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing object IDs.

If successful, OVwDbObjectIdToHostname returns a pointer to the IP host name for the object identified by objectId. If unsuccessful, it returns NULL. Because the return value for OVwDbObjectIdToHostname is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbHostnameToObjectId and OVwDbObjectIdToHostname set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_OBJECT_NOT_FOUND]	No object was found that matched either objectId or hostname.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

OVwDbHostnameToObjectId returns the following error:

[OVw_FIELD_VALUE_NULL]	The host name provided is a NULL character pointer.

Implementation Specifics

OVwDbHostNameToObjectId and OVwObjectIdToHostName support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbHostnameToObjectId or OVwDbObjectIdToHostname, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwDbNameToObjectId(3)” on page 670.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See OVwIsIdEqual in “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbInit(3)

Purpose

Initializes the OVwDb API

Syntax

```
#include <OV/ovw.h>

int OVwDbInit();
```

Description

OVwDbInit initializes internal OVwDb API data structures and the communications channel between a NetView for AIX application and the NetView for AIX database daemon, ovwdb. It must be called before any other OVwDb API call. It is called automatically by OVwInit because certain functions of the OVw APIs require some OVwDb API calls. Your application should call OVwDbInit if it is going to use only the OVwDb routines without the rest of the OVwAPI. An application using OVwDbInit does not need to be started by the NetView for AIX program.

Return Values

If successful, OVwDbInit returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwDbInit sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_ALREADY_INITIALIZED]	The API has been initialized with a prior call to OVwDbInit.
[OVw_DB_CONNECT_ERROR]	A failure occurred when attempting to connect to ovwdb.
[OVw_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbInit supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbInit, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbListFields(3)

Purpose

Lists OVW object database fields

Related Functions

OVwDbFreeFieldList

Syntax

```
#include <OV/ovw_obj.h>

OVwFieldList *OVwDbListFields(unsigned int fieldFilter);

void OVwDbFreeFieldList(OVwFieldList *fieldListBuff);
```

Description

OVwDbListFields returns a pointer to a list of fields. A field filter can be specified to determine which fields will be returned. For more information on creating fields with specific flags see “OVwDbCreateField(3)” on page 635.

OVwDbFreeFieldList frees the memory allocated for an OVwFieldList structure.

Parameters

fieldFilter Specifies a filter that indicates which fields are to be included in the return information

ovwAllFields	Returns all fields. Combining the ovwAllFields filter value with any other filter value will result in the negation of the ovwAllFields flag.
ovwListField	Returns all fields with the ovwListField flag set.
ovwNameField	Returns all fields with the ovwNameField flag set.
ovwLocateField	Returns all fields with the ovwLocateField flag set.
ovwCapabilityField	Returns all fields with the ovwCapabilityField flag set.
ovwGeneralField	Returns all fields with the ovwGeneralField flag set. Any of the above values can be used independently or by using the logical OR operator. The filter value can be combined to increase the scope of the search.

fieldListBuff Specifies a pointer to the OVwFieldList structure to be freed

Return Values

If successful, OVwDbListFields returns a pointer to an OVwFieldList structure. If unsuccessful, it returns NULL. The number of structures in the OVwFieldList may be 0 (zero), indicating no fields were found for the specified filter.

Error Codes

OVwDbListFields sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_INVALID_FILTER]	A combination of search filters that is not valid was provided.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Examples

You can combine the ovwLocateField and the ovwCapabilityField into one search filter by entering the following call:

```
OVwFieldList   *fieldlist;

if((fieldlist=OVwDbListFields(ovwLocateField | ovwCapabilityField)==NULL) {
   printf("Error: %d.\n",OVwError());
}
```

This call will cause an OVwFieldList to contain all the fields in the database that have the ovwLocateField flag set, the ovwCapabilityField flag set, or both flags set.

You can build an OVwFieldList containing every field in the OVW object database by entering the following call:

```
OVwFieldList *fieldList;

if((fieldlist=OVwDbListFields(ovwAllFields)==NULL) {
   printf("Error: %d.\n",OVwError());
}
```

Implementation Specifics

OVwDbListFields and OVwDbFreeFieldList support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbListFields or OVwDbFreeFieldList, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbGetFieldInfo(3)” on page 648.
- See “OVwDbInit(3)” on page 662.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbListObjectsByFieldValue(3)

Purpose

Lists objects by field value

Related Functions

OVwDbListObjectsByFieldValues
OVwDbFreeObjectIdList

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectIdList *OVwDbListObjectsByFieldValue(OVwFieldBinding *fieldBinding);

OVwObjectIdList *OVwDbListObjectsByFieldValues(OVwFieldBindList *fieldList);

void OVwDbFreeObjectIdList(OVwObjectIdList *objIdList);
```

Description

An object in the OVW object database is composed of a series of fields containing values. It can be useful to obtain a list of object IDs that have a specific value defined for a particular field. These functions provide the ability to search the entire OVW object database to locate objects based on field values. If the input to these functions is provided as a NULL pointer, the OVwObjectIdList returned will contain a list of every object ID in the OVW object database.

OVwDbListObjectsByFieldValue and OVwDbListObjectsByFieldValues cannot be used with fields that have the ovwListField flag set.

OVwDbListObjectsByFieldValue returns a list of objects from the OVW object database that have a single, specific value set for a field. The field ID and the value to be matched are in the structure pointed to by fieldBinding.

OVwDbListObjectsByFieldValues returns all the objects in the OVW object database that have all the field values specified by the argument fieldList. A logical AND of the fields in the list is used. That is, for an object to match, it must have all the requested fields set to their specified values.

OVwDbFreeObjectIdList frees memory allocated for an OVwObjectIdList structure.

Parameters

fieldBinding	Specifies a pointer to an OVwFieldBinding structure that contains the field ID and the field value to be used in locating objects
fieldList	Specifies a pointer to an OVwFieldBindList structure that contains a list of field IDs and their corresponding values
objIdList	Specifies the OVwObjectIdList to be freed

Return Values

If successful, OVwDbListObjectsByFieldValue returns a pointer to an OVwObjectIdList structure containing a list of object IDs for objects that have the field value indicated by fieldBinding. If no match was found, the OVwObjectIdList will contain no object IDs. If unsuccessful, OVwDbListObjectsByFieldValue returns NULL.

If successful, OVwDbListObjectsByFieldValues returns an OVwObjectIdList structure listing object IDs for all objects that match all the field values in fieldList. If no match was found, the OVwObjectIdList will contain no object IDs. If unsuccessful, OVwDbListObjectsByFieldValues returns NULL.

Error Codes

OVwDbListObjectsByFieldValue and OVwDbListObjectsByFieldValues set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
: The connection to the NetView for AIX program was lost.

[OVw_FIELD_NOT_FOUND]
: The provided field ID does not represent any field in the database.

[OVw_FIELD_TYPE_MISMATCH]
: The field type provided in an OVwFieldBinding structure does not match the field type stored in the database for the field ID.

[OVw_OUT_OF_MEMORY]
: A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
: The EUI API has not been initialized with OVwInit.

[OVw_FIELD_INVALID_FLAG]
: A field in the OVwFieldBinding has the ovwListField flag set. OVwDbListObjectsByFieldValue and its related functions do not support searches on fields that have this flag set.

Implementation Specifics

OVwDbListObjectsByFieldValue and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbListObjectsByFieldValue or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbNameToObjectId(3)

Purpose

Converts a name field to an object ID

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectId OVwDbNameToObjectId(OVwFieldId fieldId, char *nameValue);
```

Description

Setting a value for a name field creates a handle that can be used to uniquely identify any object in the OVW Object database. There is a one-to-one correspondence between the name-field value and the object ID.

OVwDbNameToObjectId enables you to translate a name-field value to an object ID.

Parameters

fieldId Specifies the ID of the name field containing the name value to be located. The fieldId must represent a field that was created with the ovwNameField field flag set. See “OVwDbCreateField(3)” on page 635.

nameValue Specifies a pointer to the name to be located in the database.

Return Values

If successful, OVwDbNameToObjectId returns the ID of the object that contains a field value matching the nameValue parameter. If unsuccessful, it returns ovwNullObjectId. The macros OVwIsIdNull and OVwIsIdEqual should be used to examine the return value.

Error Codes

OVwDbNameToObjectId sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST] The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED] An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_FIELD_NOT_FOUND] The fieldId parameter does not represent a field in the database.

[OVw_FIELD_NOT_NAME] The fieldId parameter does not represent a field that has the ovwNameField flag set.

[OVw_FIELD_VALUE_NULL] The nameValue parameter is a NULL pointer.

[OVw_OBJECT_NOT_FOUND]	There exists no object that has nameValue set for the field identified by fieldId.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbNameToObjectId supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbNameToObjectId, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateField(3)” on page 635.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbSetFieldValue(3)” on page 676.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See OVwIsIdEqual in “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbSelectionNameToObjectId(3)

Purpose

Converts an object ID to a selection name

Related Functions

OVwDbObjectIdToSelectionName

Syntax

```
#include <OV/ovw_obj.h>

OVwObjectId OVwDbSelectionNameToObjectId(char *selectionName);

char *OVwDbObjectIdToSelectionName(OVwObjectId objectId);
```

Description

Every object in the OVW database can be identified by a unique name. This name is called the selection name. The selection name for an object is defined at the time the object is created. See “OVwDbCreateObject(3)” on page 638 for more information regarding the creation of objects. OVwDbSelectionNameToObjectId and OVwDbObjectIdToSelectionName provide a convenient method of converting a selection name to an object ID and an object ID to a selection name.

OVwDbSelectionNameToObjectId returns the OVwObjectId of the object that has as a selection name matching the value provided by selectionName.

OVwDbObjectIdToSelectionName returns the selection name for the object identified by objectId.

Parameters

selectionName
Specifies a pointer to the selection name of an object.

objectId
Specifies the ID of an object.

Return Values

If successful, OVwDbSelectionNameToObjectId returns the OVwObjectId that uniquely identifies the object that has a selection name matching selectionName. If unsuccessful, it returns ovwNullObjectId. The macros OVwIsIdNull and OVwIsIdEqual should be used for testing and comparing object IDs.

If successful, OVwDbObjectIdToSelectionName returns the selection name defined for the object identified by objectId. If unsuccessful, it returns NULL. Because the return value for OVwDbObjectIdToSelectionName is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwDbSelectionNameToObjectId and OVwDbObjectIdToSelectionName set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
: The connection to ovwdb was lost.

[OVw_DB_NOT_INITIALIZED]
: The EUI API has not been initialized with OVwInit.

[OVw_DB_OPEN_FAILED]
: An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.

[OVw_OBJECT_NOT_FOUND]
: No object that matched objectId, selectionName, or hostName was found.

[OVw_OUT_OF_MEMORY]
: A memory allocation failure occurred.

OVwDbSelectionNameToObjectId returns the following error:

[OVw_FIELD_VALUE_NULL]
: The name value provided is a NULL character pointer.

Implementation Specifics

OVwDbSelectionNameToObjectId and OVwDbObjectIdToSelectionName support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbSelectionNameToObjectId or OVwDbObjectIdToSelectionName, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbInit(3)” on page 662.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwIsIdNull(3)” on page 743.
- See OVwIsIdEqual in “OVwIsIdNull(3)” on page 743.
- See “OVwApiIntro(5)” on page 560.

OVwDbSetEnumConstants(3)

Purpose

Sets values for an enumerated data type

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbSetEnumConstants(OVwFieldId fieldId, OVwEnumConstants *enumConstants);
```

Description

The range of possible values that can be set for a field that was created with the ovwEnumField field flag must be defined prior to setting any values. See “OVwDbCreateField(3)” on page 635 for more information on creating fields. The range of values is defined using an OVwEnumConstants structure.

The OVwDbSetEnumConstants routine assigns an index value to each character string listed in enumConstants. The first entry in any enumeration stored in the OVW object database must be the key word Unset. If enumConstants does not include this key word as the first value, it will be automatically added. Enumerated values are maintained in the database in the order in which they are listed in enumConstants.

Parameters

enumConstants	Specifies a pointer to an OVwEnumConstants structure.
fieldId	Identifies an object attribute field. This ID must represent a field in the OVW object database that was created with the a data type of ovwEnumField.

Return Values

If successful, OVwDbSetEnumConstants returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwDbSetEnumConstants sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NOT_FOUND]	The fieldId parameter does not identify a field in the OVW object database.
[OVw_FIELD_TYPE_MISMATCH]	The type of the field identified by fieldId is not ovwEnumField.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbSetEnumConstants supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbSetEnumConstants, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbAppendEnumConstants(3)” on page 633.
- See “OVwDbCreateField(3)” on page 635.
- See OVwDbGetFieldEnumByValue in “OVwDbGetFieldValue(3)” on page 650.
- See OVwDbGetFieldEnumByName in “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbInit(3)” on page 662.
- See OVwDbSetFieldEnumByName in “OVwDbSetFieldValue(3)” on page 676.
- OVwDbSetFieldEnumByValue in “OVwDbSetFieldValue(3)” on page 676.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbSetFieldValue(3)

Purpose

Sets a field value for an object

Related Functions

OVwDBSetFieldValues
OVwDbSetFieldBooleanValue
OVwDbSetFieldEnumByValue
OVwDbSetFieldEnumByName
OVwDbSetFieldIntegerValue
OVwDbSetFieldStringValue

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbSetFieldValues(OVwObjectId objectId,
       OVwFieldBindList *fieldBindList);

int OVwDbSetFieldValue(OVwObjectId objectId,
       OVwFieldId fieldId, OVwFieldValue *fieldValue);

int OVwDbSetFieldBooleanValue(OVwObjectId objectId,
       OVwFieldId fieldId, OVwBoolean booleanValue);

int OVwDbSetFieldEnumByName(OVwObjectId objectId,
       OVwFieldId fieldId, char *name);

int OVwDbSetFieldEnumByValue(OVwObjectId objectId,
       OVwFieldId fieldId, int value);

int OVwDbSetFieldIntegerValue(OVwObjectId objectId,
       OVwFieldId fieldId, int32 integerValue);

int OVwDbSetFieldStringValue(OVwObjectId objectId,
       OVwFieldId fieldId, char *stringValue);
```

Description

OVwDbSetFieldValue sets the value of the field identified by the fieldId parameter for the object identified by objectId. The OVwDbSetFieldValue routine can set a value for any field type.

OVwDbSetFieldValues sets the values for multiple fields identified in the fieldBindList parameter for the object identified by objectId. The OVwDbSetFieldValues routine can set values for any combination of field types. Using OVwDbSetFieldValues to set multiple fields for an object is much faster than setting fields one at a time with OVwDbSetFieldValue, because the values are batched together and sent as one request.

OVwDbSetFieldBooleanValue sets a value for a field of type ovwBooleanField.

OVwDbSetFieldEnumByName and OVwDbSetFieldEnumByValue will set a value for a field of type ovwEnumField. The field value can be set with an index into the enumerated type or an actual value

stored in the enumerated type. An error will result if the value parameter is not in the range defined for the enumerated type or if the name parameter does not exist in the type. See "OVwDbSetEnumConstants(3)" on page 674 for more information on enumerated data.

OVwDbSetFieldIntegerValue sets the value for a field of type ovwIntegerField. OVwDbSetFieldIntegerValue cannot be used to set values for fields that have the ovwListField flag set.

OVwDbSetFieldStringValue sets a value for a field of type ovwStringField. OVwDbSetFieldStringValue cannot be used to set values for fields that have the ovwListField flag set.

Parameters

booleanValue	Specifies the value to set for the boolean field
fieldId	Specifies the ID of the field to set
fieldBindList	Specifies a list of OVwFieldBindings which identify the OVwFieldIds and OVwFieldValues to set.
fieldValue	Specifies a pointer to an OVwFieldValue structure containing a field value to set
integerValue	Specifies the value to be set for the integer field
name	Specifies a pointer to the text string value to be set for the enumerated field
objectId	Specifies the object ID of the object for which the field value is to be set
stringValue	Specifies a pointer to the value to be set for the string field
value	Specifies the index number of the value to be set for the enumerated field

Return Values

If successful, OVwDbSetFieldValue and its related functions return a value of 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwDbSetFieldValue and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_DB_UPDATE_FAILURE]	An attempt to update the a field in the OVW object database failed.
[OVw_FIELD_NOT_FOUND]	The fieldId parameter does not represent a field in the OVW object database.
[OVw_FIELD_TYPE_MISMATCH]	The type of the field identified by fieldId is not consistent with the calling procedure.
[OVw_OBJECT_NOT_FOUND]	The objectId parameter does not represent an object in the OVW object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

OVwDbSetFieldEnumByValue and OVwDbSetFieldEnumByName return the following error:

[OVw_INDEX_OUT_OF_RANGE]	The input provided for the value or name are not part of the enumeration identified by fieldId.

Implementation Specifics

OVwDbSetFieldValue and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses an OVwDbSetFieldValue function, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See OVwDbGetFieldEnumByName in “OVwDbGetFieldValue(3)” on page 650.
- See OVwDbGetFieldEnumByValue in “OVwDbGetFieldValue(3)” on page 650.
- See “OVwDbGetFieldInfo(3)” on page 648.
- See “OVwDbInit(3)” on page 662.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbSetSelectionName(3)

Purpose

Sets the selection name

Related Functions

OVwDbSetHostname

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbSetSelectionName(OVwObjectId objectId, char *sname);

int OVwDbSetHostname(OVwObjectId objectId, char *hname);
```

Description

The selection name and the IP host name fields are defined by the NetView for AIX program in the object database. They are defined with the ovwNameField field flag set, so that they can be used to locate objects in the database. See “OVwDbNameToObjectId(3)” on page 670. These functions provide a convenient way to change values assigned to these fields for a given object.

OVwDbSetSelectionName resets the value of the selection name field for the particular object to the string pointed to by the sname parameter.

OVwDbSetHostname resets the value of the IP host name field for the particular object to the string pointed to by the hname parameter.

Parameters

hname
Specifies a pointer to the name to be set for the IP host name field of the object that is identified by objectId

objectId
Specifies the ID of an object in the OVW object database

sname
Specifies a pointer to the name to be set for the selection name field of the object that is identified by objectId

Return Values

If successful, OVwDbSetSelectionName and OVwDbSetHostname return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwDbSetSelectionName and OVwDbSetHostname set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_FIELD_NAME_NOT_UNIQUE]	The provided name was not unique for all values set for the field.
[OVw_FIELD_VALUE_NULL]	The provided name is a NULL character pointer.
[OVw_OBJECT_NOT_FOUND]	The objectId parameter does not represent an object in the OVW object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Implementation Specifics

OVwDbSetSelectionName and OVwDbSetHostname support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbSetSelectionName or OVwDbSetHostname, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbGetUniqObjectName(3)” on page 658.
- See “OVwDbHostnameToObjectId(3)” on page 660.
- See “OVwDbInit(3)” on page 662.
- See “OVwDbNameToObjectId(3)” on page 670.
- See “OVwDbSelectionNameToObjectId(3)” on page 672.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDbUnsetFieldValue(3)

Purpose

Unsets field values for an object

Related Functions

OVwDbUnsetFieldValues

Syntax

```
#include <OV/ovw_obj.h>

int OVwDbUnsetFieldValue(OVwObjectId objectId, OVwFieldId fieldId);

int OVwDbUnsetFieldValues(OVwObjectId objectId,
           OVwFieldIdList *fieldIdList);
```

Description

These functions provide the ability to remove fields from objects. The effects of these calls are such that upon completion, a call to OVwDbGetFieldValue with the parameters objectId and fieldId (or any field ID listed in fieldIdList) will result the error OVw_FIELD_VALUE_NULL. These calls do not affect the field definition. See “OVwDbCreateField(3)” on page 635 for more information on field definitions.

OVwDbUnsetFieldValue provides the ability to remove a field value for an object.

OVwDbUnsetFieldValues provides the ability to remove a list of field values for an object.

Parameters

fieldId
Specifies the ID of the field that contains the value to be removed

fieldIdList
Specifies a pointer to a list of IDs of fields that contain values to be removed

objectId
Specifies the ID of the object that contains the field values to be removed

Return Values

If successful, OVwDbUnsetFieldValue and OVwDbUnsetFieldValues return 1. If unsuccessful, they returns 0 (zero).

OVwDbUnsetFieldValues will fail only if objectId does not exist. If a field ID in fieldIdList does not exist, or represents a protected field, no error is returned and the function continues processing the next field ID in fieldIdList. (See the following descriptions of [OVw_FIELD_NOT_FOUND] and [OVw_FIELD_PROTECTED_VALUE].)

Error Codes

OVwDbUnsetFieldValue and OVwDbUnsetFieldValues set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_DB_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_DB_OPEN_FAILED]	An attempted connection to the OVW object database failed. When a program receives this error, it should discontinue processing until the database communication problem has been resolved.
[OVw_OBJECT_NOT_FOUND]	The objectId parameter does not represent an object in the OVW object database.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

OVwUnsetFieldValue returns the following errors:

[OVw_FIELD_NOT_FOUND]	The fieldId parameter does not represent an field in the OVW object database.
[OVw_FIELD_PROTECTED_VALUE]	An attempt was made to unset a protected field value. The field values that cannot be removed with this call include the Selection Name field, the OVW Maps Exists field, and the OVW Maps Managed field.

Implementation Specifics

OVwDbUnsetFieldValue supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDbUnsetFieldValue or OVwDbUnsetFieldValues, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovwdb(8).
- See “OVwDbInit(3)” on page 662.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDisplaySubmap(3)

Purpose

Displays a submap

Syntax

```
#include <OV/ovw.h>

int OVwDisplaySubmap(OVwMapInfo *map, OVwSubmapId submapId);
```

Description

OVwDisplaySubmap displays a submap in a submap window. If the submap is already displayed, the submap window is raised to the top of the screen.

Most map applications do not use OVwDisplaySubmap because users generally use the graphical user interface to display submaps. Only applications that create submaps as a result of interactive user requests, such as a menu operation, would need to display submaps.

Parameters

map
Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submapId
Specifies the ID of the submap to be displayed.

Return Values

If successful, OVwDisplaySubmap returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwDisplaySubmap sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap specified by submapId does not exist on the open map.

Implementation Specifics

OVwDisplaySubmap supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDisplaySubmap, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwDone(3)

Purpose

Terminates an application's connection to the NetView for AIX program

Syntax

```
#include <OV/ovw.h>

int OVwDone();
```

Description

OVwDone terminates the communications between an application and the NetView for AIX program. After calling OVwDone, the application may not make any NetView for AIX calls, except to OVwError and OVwErrorMsg. OVwDone should be called immediately before an NetView for AIX application exits.

Return Values

The OVwDone routine does not return a value; that is, it returns void.

Error Codes

OVwDone sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwDone supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwDone, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See "OVwInit(3)" on page 741.
- See "OVwError(3)" on page 688.
- See "OVwApiIntro(5)" on page 560.

OVwEndSessionCB(3)

Purpose

Functions as a callback for an end-of-session event

Syntax

```
#include <OV/ovw.h>

void (*OVwEndSessionCB) (void *userData, OVwEventType type,
          OVwBoolean normalEnd);
```

Description

To receive an event indicating that the NetView for AIX session is exiting, use OVwAddCallback(3) to register a callback function of type OVwEndSessionCB to be called when an ovwEndSession event is generated.

Note: It is recommended that every application register to receive the ovwEndSession event so that all applications can terminate correctly when the NetView for AIX program exits.

Parameters

normalEnd Indicates whether the end of the NetView for AIX session was normal. If TRUE, the session was terminated by a user request. If FALSE, the session was terminated abnormally by some signal that the NetView for AIX program received.

Examples

The following code fragment shows an example of registering a callback routine for receiving an end session event:

```
void
EndSessionProc(void *userData, OVwEventType type,
        OVwBoolean normalEnd)
{
        /* Perform application cleanup necessary before
        ** exit.
        */

        if (!normalEnd) {
                /* Do any processing necessary for an
                ** abrupt shutdown.
                */
        }

        OVwDone();
        exit(!normalEnd);
}

OVwAddCallback(ovwEndSession, NULL,
                (OVwCallbackProc) EndSessionProc, NULL);
```

Implementation Specifics

OVwEndSessionCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwEndSessionCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwError(3)

Purpose

Returns the error code set by the last EUI API call

Syntax

```
#include <OV/ovw.h>

int OVwError();
```

Description

OVwError returns the error value set by the previous EUI API call. It can be tested immediately after a failed NetView for AIX call to determine the exact reason for the failure.

Examples

The following code fragment illustrates the way OVwError can be used with OVwErrorMsg:

```
if (OVwInit() < 0) {
    fprintf(stderr, "foo: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}
```

Implementation Specifics

OVwError supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwError, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See "OVwErrorMsg(3)" on page 689.
- See "OVwInit(3)" on page 741.
- See "OVwApiIntro(5)" on page 560.

OVwErrorMsg(3)

Purpose

Returns a textual description of an EUI API error code

Syntax

```
#include <OV/ovw.h>

char *OVwErrorMsg(int error);
```

Description

OVwErrorMsg maps an EUI API error code to a string that contains text describing the meaning of the specified error code.

Parameters

error
An EUI API error code, usually the value of OVwError.

Return Values

OVwErrorMsg returns a pointer to a text string. The character array pointed to should not be modified by the program, and might be overwritten by a subsequent call to the function. Because the return value is a pointer to a static buffer, it must be copied in order to be saved.

Examples

The following code fragment illustrates the way that OVwErrorMsg can be used with OVwError:

```
if (OVwInit() < 0) {
    fprintf(stderr, "foo: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}
```

Implementation Specifics

OVwErrorMsg supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwErrorMsg, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwApiIntro(5)” on page 560.

OVwEventIntro(5)

Purpose

Provides an introduction to NetView for AIX events

Description

An application can register to receive various asynchronous events from the NetView for AIX program. This is done by registering a callback function to be called when the event occurs.

The OVwAddCallback routine can be used to register for various interesting EUI API events, such as when the NetView for AIX program exits or when a new map is opened. OVwAddCallback is called with an event type, a callback function conforming to the callback type for the indicated event type, and an optional filter based on capability field values. See “OVwMapOpenCB(3)” on page 761 for an example of registering for an EUI API event. The EUI API events and the corresponding callbacks are defined in the header file <OV/ovw.h>. Below is a summary of these events. In addition to the man pages listed below, see “OVwApiIntro(5)” on page 560 for a complete overview of the EUI API.

Table 19 (Page 1 of 2). EUI API Events and Their Callbacks

Event	Description	See Also
ovwEndSession	NetView for AIX session termination	OVwEndSessionCB
ovwSelectListChange	Map selection list changed	OVwSelectListChangeCB
ovwMapOpen	Map open	OVwMapOpenCB
ovwMapClose	Map close	OVwMapCloseCB, OVwAckMapClose
ovwQueryAppConfigChange	Application configuration change query	OVwVerifyAppConfigChange
ovwConfirmAppConfigChange	Application configuration changed	OVwVerifyAppConfigChange
ovwQueryDescribeChange	Description change query	OVwVerifyDescribeChange
ovwConfirmDescribeChange	Description changed	OVwVerifyDescribeChange
ovwQueryAddSymbol	Query to add symbol to map	OVwVerifyAdd
ovwConfirmAddSymbol	Symbol added to map	OVwVerifyAdd
ovwQueryConnectSymbols	Query to connect objects	OVwVerifyConnect
ovwConfirmConnectSymbols	Objects connected	OVwVerifyConnect
ovwQueryDeleteSymbols	Query to delete symbols	OVwVerifyDeleteSymbol
ovwConfirmDeleteSymbols	Symbols deleted from map	OVwVerifyDeleteSymbol

Table 19 (Page 2 of 2). EUI API Events and Their Callbacks

Event	Description	See Also
ovwConfirmDeleteObjects	Objects deleted from map	OVwConfirmDeleteObjectsCB
ovwConfirmDeleteSubmaps	Submaps deleted from map	OVwConfirmDeleteSubmapsCB
ovwConfirmCreateSymbols	Symbols created on map	OVwConfirmCreateSymbolsCB
ovwConfirmCreateObjects	Objects created on map	OVwConfirmCreateObjectsCB
ovwConfirmCreateSubmaps	Submaps created on map	OVwConfirmCreateSubmapsCB
ovwConfirmMoveSymbol	Symbol moved	OVwConfirmMoveSymbolCB
ovwConfirmManageObjects	Objects managed	OVwConfirmManageObjectsCB
ovwConfirmUnmanageObjects	Objects unmanaged	OVwConfirmManageObjectsCB
ovwConfirmHideSymbols	Symbols hidden	OVwConfirmHideSymbolsCB
ovwConfirmUnhideSymbols	Symbols unhidden	OVwConfirmHideSymbolsCB
ovwConfirmSymbolStatusChange	Symbol status change	OVwConfirmObjectStatusCB
ovwConfirmObjectStatusChange	Object status change	OVwConfirmObjectStatusCB
ovwConfirmCompoundStatusChange	Compound object status change	OVwConfirmObjectStatusCB
ovwConfirmCapabilityChange	Object capability field change	OVwConfirmCapabilityChangeCB
ovwConfirmAcknowledgeObjects	Object acknowledged	OVwConfirmAcknowledgeObjectsCB
ovwConfirmUnacknowledgeObjects	Object unacknowledged	OVwConfirmAcknowledgeObjectsCB
ovwConfirmExplodeObject	Object exploded to display child submap	OVwConfirmExplodeObjectCB
ovwUserSubmapCreate	Submap creation notification	OVwUserSubmapCreateCB, OVwAckUserSubmapCreate
ovwSubmapOpen	Submap open	OVwSubmapOpenCB
ovwSubmapClose	Submap close	OVwSubmapCloseCB

Libraries

When compiling a program that uses any of the callbacks corresponding to NetView for AIX events, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwFileDescriptor(3)

Purpose

Accesses the NetView for AIX program communications channel

Syntax

```
#include <OV/ovw.h>

int OVwFileDescriptor()
```

Description

OVwFileDescriptor is a macro that returns the file descriptor associated with the socket connecting the application to the NetView for AIX program. Among other things, it could be used by an application's own select(2) to determine if there is input from the the NetView for AIX program session to process.

Return Values

If successful, OVwFileDescriptor returns a non-negative integer. If unsuccessful, it returns –1 (negative one).

Examples

The following fragment from OVwXtAddInput illustrates how to use OVwFileDescriptor.

Note: There can be outstanding NetView for AIX events to process even if there is a lack of available input on the file descriptor. Any application-specific select(2) processing should use OVwPending and OVwProcessEvent as they are used in the following example.

```
void __OVwXtInputCB(XtPointer closure, int *fd, XtInputId *id)
{
    /* Flush any pending ovw events */
    while (OVwPending()) {
        if (OVwProcessEvent() < 0) {
            XtRemoveInput(*id);
            OVwDone();
            return;
        }
    }
}

XtInputId OVwXtAddInput()
{
    XtInputId id;
    int fd;

    fd = OVwFileDescriptor();
    if (fd >= 0)
        id = XtAddInput (fd, (XtPointer)XtInputReadMask, __OVwXTInputCB, NULL);
    return id;
}
```

Implementation Specifics

OVwFileDescriptor supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwFileDescriptor, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwInit(3)” on page 741.
- See “OVwPending(3)” on page 765.
- See “OVwProcessEvent(3)” on page 767.
- See “OVwApiIntro(5)” on page 560.

OVwFindMenuItem(3)

Purpose

Finds a menu item

Related Functions

OVwFindObjMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwFindMenuItem(char *menuItemPath);

char *OVwFindObjMenuItem(char *menuItemPath);
```

Description

OVwFindMenuItem returns a pointer to the ID of the menu item in the current registration context that is located at the specified path in the graphical interface menu bar.

OVwFindObjMenuItem returns a pointer to the ID of the menu item in the current registration context that is located at the specified path in the graphical interface menu bar.

Parameters

menuItemPath
Specifies a pointer to a string that specifies a location in the menu bar. This location consists of the labels on the graphical interface of the cascades and buttons for the menu item, separated by the current menu path separator. See OVwSetMenuPathSeparator in “OVwGetMenuPathSeparator(3)” on page 723. The default menu path separator is →.

Return Values

If successful, OVwFindMenuItem and OVwFindObjMenuItem return a non-NULL character pointer. If unsuccessful, it returns a character pointer. Because the return value for these functions is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwFindMenuItem and OVwFindObjMenuItem set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]
The connection to the NetView for AIX program was lost.

[OVw_MENUITEM_NOT_FOUND]
The current registration context does not have a menu item registered for the specified menu path.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The EUI API has not been initialized with OVwInit.

Examples

You can use the following code fragment to locate the ID of a menu item:

```
char *id = OVwFindMenuItem("Administer→Telnet (aixterm)...");
if (id == NULL) {
        fprintf(stderr, "error: %s\n", OVwErrorMsg(OVwError()));
        return -1;
}
```

Implementation Specifics

OVwFindMenuItem and OVwFindObjMenuItem support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwFindMenuItem, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwCreateMenuItem(3)” on page 613.
- See OVwSetMenuPathSeparator in “OVwGetMenuPathSeparator(3)” on page 723.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetAppConfigValues(3)

Purpose

Returns application configuration parameters

Related Functions

OVwSetAppConfigValues

Syntax

```
#include <OV/ovw.h>

OVwFieldBindList *OVwGetAppConfigValues(OVwMapInfo *map,
       char *appName);

int OVwSetAppConfigValues(OVwMapInfo *map,
       OVwFieldBindList *configParams);
```

Description

OVwGetAppConfigValues returns a pointer to a list of fields that have the current values of the map application-configuration parameters for each map for the specified application. The application configuration parameters for each map are specified in the application registration file.

OVwSetAppConfigValues takes a pointer to a list of fields and their values and sets the values accordingly in the database. The fields specified in this call must be configuration fields as defined in the application registration file for the application making that call. To set the values of object fields use OVwSetFieldValues.

OVwSetAppConfigValues is useful when a particular configuration parameter can be set either from the command line or the Configuration dialog box. In this case, the application can place the values retrieved from the command line directly into the database. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API, including the role of the Configuration dialog box.

Parameters

appName Specifies a pointer to the name of the application for which the configuration parameters are needed. If NULL, OVwGetConfigValues returns the configuration parameters for the application making the call.

configParams Specifies a pointer to a list of configuration parameters whose values are to be set in the database for the application making the call.

map Specifies a pointer to a MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

Return Values

If successful, OVwGetAppConfigValues returns a pointer to an OVwFieldBindList structure. If unsuccessful, it returns a NULL pointer.

If successful, OVwSetAppConfigValues returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwGetAppConfigValues and OVwSetAppConfigValues set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND] The application appName is not a registered application.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The EUI API has not been initialized with OVwInit.

Examples

The following example illustrates how to check the application's configuration field values:

```
OVwFieldBindList  *fieldListPtr;
OVwMapInfo        *map = OVwGetMapInfo();

/*
** Check the values of the fields and set up accordingly.
*/

fieldListPtr = OVwGetAppConfigValues (map, NULL);
if (fieldListPtr == NULL) {
    /* bail out */
} else {
    /* walk list and get the field values */

    /* now free the list */

    OVwDbFreeFieldBindList (fieldListPtr);

}
    OVwFreeMapInfo(map);
```

Implementation Specifics

OVwGetAppConfigValues and OVwSetAppConfigValues support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetAppConfigValues or OVwSetAppConfigValues, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetAppName(3)

Purpose

Returns the name of the running application

Syntax

```
#include <OV/ovw.h>

char *OVwGetAppName();
```

Description

OVwGetAppName returns the name of the currently running application as it was registered with the NetView for AIX program. This name is used in other NetView for AIX calls to uniquely identify the application.

Return Values

If successful, OVwGetAppName returns a pointer to a dynamically allocated string containing the name of the application. If unsuccessful, OVwGetAppName returns NULL.

Because the return value is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetAppName set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwGetAppName supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetAppName, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwGetConnSymbol(3)

Purpose

Returns a connection symbol

Syntax

```
#include <OV/ovw.h>

OVwSymbolInfo *OVwGetConnSymbol(OVwMapInfo *map,
       OVwSymbolId endpoint1, OVwSymbolId endpoint2);
```

Description

OVwGetConnSymbol returns a pointer to symbol info for the connection symbol that connects the two symbols endpoint1 and endpoint2, if such a symbol exists.

The SymbolInfo structure returned by OVwGetConnSymbol may be for a metaconnection symbol that represents multiple connections between the two symbols. This is indicated by the is_meta_conn field of the OVwSymbolInfo structure. OVwListSymbols can be used to get the connections represented by a meta-connection symbol.

OVwFreeSymbolInfo should be used to free the OVwSymbolInfo structure returned by OVwGetConnSymbol. See “OVwGetSymbolInfo(3)” on page 735 for more information about OVwFreeSymbolInfo.

Parameters

endpoint1	Specifies the symbol ID of an icon symbol that is a connection end point.
endpoint2	Specifies the symbol ID of an icon symbol that is a connection end point. The special value ovwSubmapBackbone can be used if the layout of the submap (ovwBusLayout or ovwRingLayout) has a backbone.
map	Specifies a pointer to a map structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

Return Values

If successful, OVwGetConnSymbol returns a pointer to an OVwSymbolInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwGetConnSymbol sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_CONN_SYMBOL_BOTH_ENDS_NULL]	Both connection end points have the value ovwNullSymbolId.

[OVw_CONN_SYMBOL_BOTH_ENDS_SAME]	Both connection end points are the same.
[OVw_CONN_SYMBOL_END_NOT_FOUND]	One of the connection end point symbols does not exist on the open map.
[OVw_CONN_SYMBOL_END_WRONG_VARIETY]	The variety of a connection end point symbol is valid. Only icon symbols are allowed as connection end points.
[OVw_CONN_SYMBOL_ENDS_DIFFERENT_SUBMAPS]	The two connection end points are on different submaps.
[OVw_CONN_SYMBOL_NO_SUBMAP_BACKBONE]	A value of ovwSubmapBackbone is specified as a connection end point for a submap that does not have a backbone.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SYMBOL_NOT_FOUND]	The two symbols endpoint1 and endpoint2 are not connected by any connection symbol.

Examples

The following code fragment shows how to get a connection symbol:

```
int i;
OVwSymbolInfo *syminfo;
OVwSymbolList *symlist;
OVwMapInfo *map = OVwGetMapInfo();

syminfo = OVwGetConnSymbol(map, end1_id, end2_id);
if (!syminfo) {
    printf("No connection!\n");
}
else if (!syminfo->conn_is_meta_conn) {
    printf("Single connection.\n");
    /* symbol represents the object syminfo->object */
}
else {
    printf("Meta-connection.\n");
    /* get symbols on meta-connection submap */
    symlist = OVwListSymbols(map,
        syminfo->object.child_submap_id, ovwAllPlanes, NULL);
    if (symlist) {
        for (i = 0; i < symlist->count; i++) {
            if (symlist->symbols[i].symbol_variety ==
                    ovwConnSymbol) {
                printf("Connection found.\n");
                /* symlist->symbols[i].object is object */
            }
        }
        OVwFreeSymbolList(symlist);
    }
}

if (syminfo)
    OVwFreeSymbolInfo(syminfo);
OVwFreeMapInfo(map);
```

Implementation Specifics

OVwGetConnSymbol supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetConnSymbol, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwGetSymbolInfo(3)” on page 735.
- See “OVwInit(3)” on page 741.
- See “OVwListSymbols(3)” on page 750.
- See “OVwApiIntro(5)” on page 560.

OVwGetFirstAction(3)

Purpose

Gets registered application actions

Related Functions

OVwGetNextAction

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetFirstAction();

char *OVwGetNextAction();
```

Description

OVwGetFirstAction and OVwGetNextAction are used to get the names of all registration actions for the current registration context. The registration context is the application for which subsequent registration calls, such as OVwGetApp, are effective. For more information about the registration context, see “OVwGetRegContext(3)” on page 729.

OVwGetFirstAction returns a pointer to the name of the first action registered in the current registration context. It should be called before OVwGetNextAction to restart the name traversal.

OVwGetNextAction returns a pointer to the name of the next action registered in the current registration context. It should be called repeatedly until it returns NULL, indicating that all action names have been returned.

Return Values

If successful, OVwGetFirstAction and OVwGetNextAction return a non-NULL pointer. If unsuccessful, they return NULL. Because the return value for OVwGetFirstAction and OVwGetNextAction is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetFirstAction and OVwGetNextAction set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUCCESS]	All action names have been obtained.

Examples

You can get all actions registered for the current registration context, retrieving and processing the details of the registered action by entering the following code:

```
char *action;
OVwActionRegInfo *info;

for (action = OVwGetFirstAction(); action; action = OVwGetNextAction()) {
    info = OVwGetAction(action);
    if (!info) {
        fprintf(stderr, "Error: %s\n", OVwErrorMsg(OVwError()));
        return;
    }
    /* process action information */
    printf("Action: %s\n", action);
    printf("SelectionRule: %s\n", info->selection_rule);

    OVwFreeActionRegInfo(info);
}
```

Implementation Specifics

OVwGetFirstAction and OVwGetNextAction support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstAction or OVwGetNextAction, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwCreateAction(3)” on page 603.
- See OVwGetAction in “OVwCreateAction(3)” on page 603.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetFirstMenuItem(3)

Purpose

Gets registered menu items

Related Functions

OVwGetNextMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetFirstMenuItem(char *menuitem, int function, char *fnArg);

char *OVwGetNextMenuItem();
```

Description

OVwGetFirstMenuItem and OVwGetNextMenuItem are used to get the IDs (names) of menu items, except for object menu items, registered in the current registration context.

OVwGetFirstMenuItem returns the ID of the first menu item registered in the current registration context. The registration context is the application for which subsequent registration calls, such as OVwGetApp, are effective. For more information about the registration context, see “OVwGetRegContext(3)” on page 729.

OVwGetFirstMenuItem should be called before OVwGetNextMenuItem to restart the traversal. The function and fnArg parameters allow traversal of subsets of menu items. Menu-item subsets include those with a particular function type, such as all menu items with actions bound to them, and those with a specific function, such as all menu items with action *Foo* bound to them.

OVwGetNextMenuItem returns the next menu item ID registered in the current registration context. It should be called repeatedly until it returns NULL, indicating that all menu item IDs have been iterated.

Parameters

fnArg Specifies a pointer to the function argument whose meaning is determined by the function parameter. If fnArg is NULL, all menu items with the specified function have been obtained.

function Specifies the type of function bound to the menu item. If function is 0 (zero), all menu items are returned. Otherwise, only those menu items with the specified type of function and function argument are returned. The function types are defined in the <OV/ovw.h> header file as follows. For each of the following function types, the parameters menuitem, function, and fnArg are passed on the call, and the return value is set by the call.

ovwMenu The fnArg function argument is a menu identifier. For example, if fnArg is “IP Commands”, this would be equivalent to specifying f.menu “IP Commands” for the menu item in the application registration file.

ovwInternal	The fnArg function argument is an internal function name. For example, if fnArg is “exit”, this would be equivalent to specifying f.exit for the menu item in the application registration file.
ovwAction	The fnArg function argument is an action identifier. For example, if fnArg is “Get”, this would be equivalent to specifying f.action “Get” for the menu item in the application registration file.
ovwShell	The fnArg function argument is a shell command. For example, if fnArg is “xterm -e /etc/ping ${OVwSelection1}”, this would be equivalent to specifying “! “xterm -e /etc/ping ${OVwSelection1}” for the menu item in the application registration file.

Return Values

If successful, OVwGetFirstMenuItem and OVwGetNextMenuItem return a non-NULL pointer. If unsuccessful, they return NULL. Because the return value for OVwGetFirstMenuItem and OVwGetNextMenuItem is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetFirstMenuItem and OVwGetNextMenuItem set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUCCESS]	All menu item IDs have been obtained.

Implementation Specifics

OVwGetFirstMenuItem and OVwGetNextMenuItem support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstMenuItem or OVwGetNextMenuItem, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetFirstMenuItemFunction(3)

Purpose

Gets functions bound to a menu item

Related Functions

OVwGetNextMenuItemFunction

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwGetFirstMenuItemFunction(char *menuItemId,
       int *function, char **fnArg);

int OVwGetNextMenuItemFunction(int *function,
       char **fnArg);
```

Description

OVwGetFirstMenuItemFunction and OVwGetNextMenuItemFunction are used to get all the functions bound to a registered menu item in the current registration context. These routines do not get the functions bound to a registered object menu item. A menu item function consists of an integer-function type and a character-string function argument.

OVwGetFirstMenuItemFunction returns the type and argument for the first function bound to the specified menu item in the current registration context. It should be called before OVwGetNextMenuItemFunction to restart the function traversal.

OVwGetNextMenuItemFunction returns the next function bound to the specified menu item in the current registration context. It should be called repeatedly until it returns NULL, indicating that all application names have been iterated.

Parameters

fnArg — If non-NULL, the contents of the pointer are set to a string that is the function argument whose meaning is determined by the function parameter.

function — If non-NULL, the contents of the pointer are set to the integer value that specifies the type of function bound to the menu item. The function types are defined in the OV/ovw.h header file as follows:

- ovwMenu — The fnArg function argument is a menu identifier. For example, if fnArg is “IP Commands”, this would be equivalent to specifying f.menu IP Commands for the menu item in the application registration file.
- ovwInternal — The fnArg function argument is an internal function name. For example, if fnArg is “exit”, this would be equivalent to specifying f.exit for the menu item in the application registration file.

ovwAction — The fnArg function argument is an action identifier. For example, if fnArg is “Get”, this would be equivalent to specifying f.action “Get” for the menu item in the application registration file.

ovwShell — The fnArg function argument is a shell command. For example, if fnArg is “xterm -e /etc/ping ${OVwSelection1}”, this would be equivalent to specifying ! “xterm -e /etc/ping ${OVwSelection1}” for the menu item in the application registration file.

menuItemId — A pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindMenuItem.

Return Values

If successful, OVwGetFirstMenuItemFunction and OVwGetNextMenuItemFunction return a non-NULL pointer. If unsuccessful, they return NULL.

Error Codes

OVwGetFirstMenuItemFunction and OVwGetNextMenuItemFunction set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] — The connection to the NetView for AIX program was lost.

[OVw_MENUITEM_NOT_FOUND] — The specified menuItemId does not refer to a menu item in the current registration context.

[OVw_OUT_OF_MEMORY] — A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] — The EUI API has not been initialized with OVwInit.

[OVw_SUCCESS] — All menu item functions have been obtained.

Implementation Specifics

OVwGetFirstMenuItemFunction and OVwGetNextMenuItemFunction support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstMenuItemFunction or OVwGetNextMenuItemFunction, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetFirstObjMenuItem(3)

Purpose

Gets registered object menu items

Related Functions

OVwGetNextObjectMenuItem

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetFirstObjMenuItem(char *menuid, int function, char *fnArg);

char *OVwGetNextObjMenuItem();
```

Description

OVwGetFirstObjMenuItem and OVwGetNextObjMenuItem are used to get the IDs (names) of object menu items registered in the current registration context.

OVwGetFirstObjMenuItem returns the ID of the first object menu item registered in the current registration context. The registration context is the application for which subsequent registration calls, such as OVwGetApp, are effective. For more information about the registration context, see “OVwGetRegContext(3)” on page 729.

OVwGetFirstObjMenuItem should be called before OVwGetNextObjMenuItem to restart the traversal. The function and fnArg parameters allow traversal of subsets of object menu items. Menu-item subsets include those with a particular function type, such as all menu items with actions bound to them, and those with a specific function, such as all menu items with action *Foo* bound to them.

OVwGetNextObjMenuItem returns the next menu item ID registered in the current registration context. It should be called repeatedly until it returns NULL, indicating that all menu item IDs have been iterated.

Parameters

menuId	Specifies a pointer to a menu identifier if you are interested in object menu items from a specified menu. If menuId is NULL, the first object menu item for the application is returned.
fnArg	Specifies a pointer to the function argument whose meaning is determined by the function parameter. If fnArg is NULL, all menu items with the specified function have been obtained.
function	Specifies the type of function bound to the menu item. If function is 0 (zero), all menu items are returned. Otherwise, only those menu items with the specified type of function and function argument are returned. The function types are defined in the <OV/ovw.h> header file as follows. For each of the following function types, the parameters menuitem, function, and fnArg are passed on the call, and the return value is set by the call.

ovwMenu	The fnArg function argument is a menu identifier. For example, if fnArg is “IP Commands”, this would be equivalent to specifying f.menu “IP Commands” for the menu item in the application registration file.
ovwInternal	The fnArg function argument is an internal function name. For example, if fnArg is “exit”, this would be equivalent to specifying f.exit for the menu item in the application registration file.
ovwAction	The fnArg function argument is an action identifier. For example, if fnArg is “Get”, this would be equivalent to specifying f.action “Get” for the menu item in the application registration file.
ovwShell	The fnArg function argument is a shell command. For example, if fnArg is “xterm -e /etc/ping ${OVwSelection1}”, this would be equivalent to specifying “! “xterm -e /etc/ping ${OVwSelection1}” for the menu item in the application registration file.

Return Values

If successful, OVwGetFirstObjMenuItem and OVwGetNextObjMenuItem return a non-NULL pointer. If unsuccessful, they return NULL. Because the return value for OVwGetFirstObjMenuItem and OVwGetNextObjMenuItem is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetFirstObjMenuItem and OVwGetNextObjMenuItem set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUCCESS]	All menu item IDs have been obtained.

Implementation Specifics

OVwGetFirstObjMenuItem and OVwGetNextObjMenuItem support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstObjMenuItem or OVwGetNextObjMenuItem, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetFirstObjMenuItemFunction(3)

Purpose

Gets functions bound to an object menu item

Related Functions

OVwGetNextObjMenuItemFunction

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwGetFirstObjMenuItemFunction(char *menuItemId,
       int *function, char **fnArg);

int OVwGetNextObjMenuItemFunction(int *function,
       char **fnArg);
```

Description

OVwGetFirstObjMenuItemFunction and OVwGetNextObjMenuItemFunction are used to get all the functions bound to a registered object menu item in the current registration context. A menu item function consists of an integer-function type and a character-string function argument.

OVwGetFirstObjMenuItemFunction returns the type and argument for the first function bound to the specified object menu item in the current registration context. It should be called before OVwGetNextObjMenuItemFunction to restart the function traversal.

OVwGetNextObjMenuItemFunction returns the next function bound to the specified object menu item in the current registration context. It should be called repeatedly until it returns NULL, indicating that all application names have been iterated.

Parameters

fnArg — If non-NULL, the contents of the pointer are set to a string that is the function argument whose meaning is determined by the function parameter.

function — If non-NULL, the contents of the pointer are set to the integer value that specifies the type of function bound to the menu item. The function types are defined in the OV/ovw.h header file as follows:

- ovwMenu — The fnArg function argument is a menu identifier. For example, if fnArg is “IP Commands”, this would be equivalent to specifying f.menu IP Commands for the menu item in the application registration file.
- ovwInternal — The fnArg function argument is an internal function name. For example, if fnArg is “exit”, this would be equivalent to specifying f.exit for the menu item in the application registration file.
- ovwAction — The fnArg function argument is an action identifier. For example, if fnArg is “Get”, this would be equivalent to specifying f.action “Get” for the menu item in the application registration file.

ovwShell	The fnArg function argument is a shell command. For example, if fnArg is “xterm -e /etc/ping ${OVwSelection1}”, this would be equivalent to specifying ! “xterm -e /etc/ping ${OVwSelection1}” for the menu item in the application registration file.
menuItemId	A pointer to a menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindObjMenuItem.

Return Values

If successful, OVwGetFirstObjMenuItemFunction and OVwGetNextObjMenuItemFunction return a non-NULL pointer. If unsuccessful, they return NULL.

Error Codes

OVwGetFirstObjMenuItemFunction and OVwGetNextObjMenuItemFunction set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENUITEM_NOT_FOUND]	The specified menuItemId does not refer to an object menu item in the current registration context.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUCCESS]	All menu item functions have been obtained.

Implementation Specifics

OVwGetFirstObjMenuItemFunction and OVwGetNextObjMenuItemFunction support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstObjMenuItemFunction or OVwGetNextObjMenuItemFunction, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetFirstRegContext(3)

Purpose

Gets registered applications

Related Functions

OVwGetNextRegContext

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetFirstRegContext(char *parentAppName);

char *OVwGetNextRegContext();
```

Description

OVwGetFirstRegContext and OVwGetNextRegContext are used to get all NetView for AIX registration contexts, that is, the name of each application that is registered with the NetView for AIX program.

OVwGetFirstRegContext returns the name of the first application in the NetView for AIX program's list of registered applications. It should be called before OVwGetNextRegContext to restart the name traversal.

OVwGetNextRegContext returns the next registration context name in the NetView for AIX program's list of registered applications. It should be called repeatedly until it returns NULL, indicating that all application names have been returned.

Parameters

parentAppName
Specifies a pointer to the name of an application that is the parent for child applications. If a parentAppName is supplied, only that application's children are returned. If parentAppName is NULL, all applications are returned.

Return Values

If successful, OVwGetFirstRegContext and OVwGetNextRegContext return a non-NULL pointer. If unsuccessful, they return NULL. Because the return value for OVwGetFirstRegContext and OVwGetNextRegContext is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetFirstRegContext and OVwGetNextRegContext set an error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUCCESS]	The iteration of application names is complete.

Examples

The following code fragment illustrates how to get all application names and print some registration information for each application:

```
OVwAppRegInfo *appInfo;
char *context;
char *savedContext = OVwGetRegContext();

for (context = OVwGetFirstRegContext(NULL); context;
     context = OVwGetNextRegContext()) {
    OVwSetRegContext(context);
    appInfo = OVwGetApp();
    if (!appInfo) {
        fprintf(stderr, "Error: %s\n", OVwErrorMsg(OVwError()));
        exit(1);
    }
    /* process application info */
    printf("Application: %s\n", context);
    printf("Version: %s\n", appInfo->version);
    printf("Command: %s\n", appInfo->command);
    OVwFreeAppRegInfo(appInfo);
}
OVwSetRegContext(savedContext);
```

Implementation Specifics

OVwGetFirstRegContext supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetFirstRegContext or OVwGetNextRegContext, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetRegContext(3)” on page 729.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.
- See OVwGetApp in “OVwCreateApp(3)” on page 607.

OVwGetMapInfo(3)

Purpose

Returns map information

Related Functions

OVwCopyMapInfo
OVwFreeMapInfo

Syntax

```
#include <OV/ovw.h>

OVwMapInfo *OVwGetMapInfo();

OVwMapInfo *OVwCopyMapInfo(OVwMapInfo *map);

void OVwFreeMapInfo(OVwMapInfo *map);
```

Description

OVwGetMapInfo returns information about the open map. OVwGetMapInfo should be called by a map application when it starts to get information about the open map. because an ovwMapOpen event is not generated for the first map opened at startup. A map application will be notified when a new map is opened by registering for an ovwMapOpen event. See “OVwMapOpenCB(3)” on page 761.

OVwCopyMapInfo allocates memory for an OVwMapInfo structure and returns a pointer to a copy of the specified map structure. This can be used in the callback for the ovwMapOpen event to save the map parameter for use in subsequent calls that deal with the open map.

OVwFreeMapInfo frees the memory allocated for an OVwMapInfo structure. It should be used to free the OVwMapInfo structure returned by OVwGetMapInfo when it is no longer needed.

Parameters

map Specifies a pointer to an OVwMapInfo structure to free or copy

Return Values

If successful, OVwGetMapInfo and OVwCopyMapInfo return a pointer to an OVwMapInfo structure. If unsuccessful, they return NULL.

Error Codes

OVwGetMapInfo and OVwFreeMapInfo set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

OVwGetMapInfo might return the following additional errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	There is no open map.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwGetMapInfo and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetMapInfo or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwMapOpenCB(3)” on page 761.
- See “OVwApiIntro(5)” on page 560.

OVwGetMenuItemPath(3)

Purpose

Retrieves location information for menu items

Related Functions

OVwGetMenuItemMenu

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

 char *OVwGetMenuItemPath(char *menuItemId);

 char *OVwGetMenuItemMenu(char *menuItemId);
```

Description

OVwGetMenuItemPath returns the path that specifies the location of the menu item in the graphical interface menu bar structure. The string consists of the labels of the cascades and buttons for the menu item, separated by the current menu path separator. The default menu path separator is →. This function is the converse of OVwFindMenuItem.

OVwGetMenuItemMenu returns the ID of the menu to which the item is attached. A menu item is uniquely identified by this menu ID and its label, so a menu item can be attached to a maximum of one menu.

Parameters

menuItemId	Specifies a pointer to menu item identifier returned from an OVwMenuItemRegistration call or from OVwFindMenuItem

Return Values

If successful, OVwGetMenuItemPath and OVwGetMenuItemMenu return a non-NULL pointer. If unsuccessful, they return a NULL pointer. Because the return value for OVwGetMenuItemPath and OVwGetMenuItemMenu is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetMenuItemPath and OVwGetMenuItemMenu set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENUITEM_NOT_FOUND]	The argument menuItemId does not specify a menu item registered in the current registration context.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwGetMenuItemPath and OVwGetMenuItemMenu support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetMenuItemPath or OVwGetMenuItemMenu, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwFindMenuItem(3)” on page 696.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwGetObjectMenuItemPath(3)” on page 727.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetMenuPathSeparator(3)

Purpose

Gets menu path separator string

Related Functions

OVwSetMenuPathSeparator

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetMenuPathSeparator();

int OVwSetMenuPathSeparator(char *separator);
```

Description

OVwSetMenuPathSeparator sets to the specified value the character string used to separate menu labels in a string that represents the path of a menu item on the menu bar. The default menu separator string is →.

OVwGetMenuPathSeparator returns the current string used to separate menu labels in a menu path string.

These routines affect how parameters to other registration routines behave by changing the way menu path names are expressed. See “OVwFindMenuItem(3)” on page 696 and OVwGetMenuItemMenu in “OVwGetMenuItemPath(3)” on page 721. These changes affect the way menu path names are expressed for the NetView for AIX program, but not for other applications.

Parameters

separator Specifies a pointer to a character string to be used as the menu path separator. The separator should be set to a string that does not appear in menu labels. It should be limited to nonalphabetic, printable characters.

Return Values

If successful, OVwGetMenuPathSeparator returns a non-NULL character pointer. If unsuccessful, it returns a NULL character pointer. Because the return value is dynamically allocated, you must free the string when it is no longer needed.

If successful, OVwSetMenuPathSeparator returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwGetMenuPathSeparator and OVwSetMenuPathSeparator set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

```
OVwSetMenuPathSeparator("==");

char *id = OVwFindMenuItem("Administer.==Telnet (aixterm)");

if (id == NULL) {
        fprintf(stderr, "error: %s\n", OVwErrorMsg(OVwError()));
        return -1;
}
```

Implementation Specifics

OVwGetMenuPathSeparator and OVwSetMenuPathSeparator support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetMenuPathSeparator or OVwSetMenuPathSeparator, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetObjectInfo(3)

Purpose

Returns map-specific object information

Related Functions

OVwFreeObjectInfo

Syntax

```
#include <OV/ovw.h>

OVwObjectInfo *OVwGetObjectInfo(OVwMapInfo *map, OVwObjectId objectId);

void OVwFreeObjectInfo(OVwObjectInfo *object);
```

Description

OVwGetObjectInfo returns information about an object on the open map. The information returned in the OVwObjectInfo structure is map-specific, except for object_id and field_values (the latter is set only in certain cases).

If OVwGetObjectInfo returns NULL and OVwError returns the error code [OVw_OBJECT_NOT_ON_MAP], the object does not exist on the open map. The object might still exist in the OVW object database. Use OVwDbObjectIdToSelectionName to determine whether the object identified by objectId exists in the OVW object database.

If the child_submap_id field of the OVwObjectInfo structure is ovwNullSubmapId, the object has no child submap on the open map; otherwise, child_submap_id specifies the submap ID of the child submap of the object on the open map.

OVwFreeObjectInfo frees memory allocated for an OVwObjectInfo structure. It should be used to free the OVwObjectInfo structure returned by OVwGetObjectInfo.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
object	Specifies a pointer to the OVwObjectInfo structure to free.
objectId	Specifies the object ID of the object.

Return Values

If successful, OVwGetObjectInfo returns a pointer to an OVwObjectInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwGetObjectInfo sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OBJECT_NOT_ON_MAP]	The object does not exist on the open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwGetObjectInfo supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetObjectInfo or OVwFreeObjectInfo, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwDbCreateObject(3)” on page 638.
- See “OVwDbSelectionNameToObjectId(3)” on page 672.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwGetObjectMenuItemPath(3)

Purpose

Retrieves location information for object menu items

Related Functions

OVwGetObjectMenuItemMenu

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

 char *OVwGetObjMenuItemPath(char *objMenuItemId);

 char *OVwGetObjMenuItemMenu(char *objMenuItemId);
```

Description

OVwGetObjMenuItemPath returns the path that specifies the location of the object menu item in the object menu structure. The string consists of the labels of the cascades and the buttons for the menu item. The default menu path separator is →. This function is the converse of OVwFindObjMenuItem.

OVwGetObjMenuItemMenu returns the ID of the menu to which the item is attached. A menu item is uniquely identified by this menu ID and its label, so a menu item can be attached to a maximum of one menu.

Parameters

objMenuItemId	Specifies a pointer ton object menu item identifier returned from OVwFindObjMenuItem

Return Values

If successful, OVwGetObjMenuItemPath and OVwGetObjMenuItemMenu return a non-NULL pointer. If unsuccessful, they return a NULL pointer. Because the return value for OVwGetObjMenuItemPath and OVwGetObjMenuItemMenu is dynamically allocated, you must free the string when it is no longer needed.

Error Codes

OVwGetObjMenuItemPath and OVwGetObjMenuItemMenu set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MENUITEM_NOT_FOUND]	The argument menuItemId does not specify a menu item registered in the current registration context.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwGetObjMenuItemPath and OVwGetObjMenuItemMenu support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetObjMenuItemPath or OVwGetObjMenuItemMenu, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwFindMenuItem(3)” on page 696.
- See “OVwCreateMenuItem(3)” on page 613.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetRegContext(3)

Purpose

Retrieves the application registration context

Related Functions

OVwSetRegContext

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

char *OVwGetRegContext();

int OVwSetRegContext(char *appName);
```

Description

OVwGetRegContext returns the name of the current registration context, which, if the application has previously called OVwSetRegContext, can be different from the application that is making the call.

OVwSetRegContext sets the registration context to that of the specified application.

The registration context is the application for which subsequent registration calls, such as OVwGetApp, are effective. For example, if the registration context were Foo, a call to OVwGetApp would retrieve the registration information for the application Foo, not that of the NetView for AIX application that is making the call.

The current registration context defaults to the application that is running. Use OVwGetAppName to retrieve the name of the application making the call.

Parameters

appName
Specifies an application name as it appears in the application registration file. If appName is NULL, OVwSetRegContext sets the current registration context to the application that is making the call.

Return Values

If successful, OVwGetRegContext returns a non-NULL character pointer. If unsuccessful, it returns NULL. Because the return value for OVwGetRegContext is dynamically allocated, you must free the string when it is no longer needed.

If successful, OVwSetRegContext returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwGetRegContext and OVwSetRegContext set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	The application appName is not a registered application.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following code shows how the routines can be used to retrieve application registration information for NetView for AIX applications. It iterates all applications and prints some registration information for each application:

```
OVwAppRegInfo *appInfo;
char *context;
char *savedContext = OVwGetRegContext();

for (context = OVwGetFirstRegContext(NULL); context;
     context = OVwGetNextRegContext()) {
    OVwSetRegContext(context);
    appInfo = OVwGetApp();
    if (!appInfo) {
        fprintf(stderr, "Error: %s\n", OVwErrorMsg(OVwError()));
        exit(1);
    }
    /* process application info */
    printf("Application: %s\n", context);
    printf("Version: %s\n", appInfo->version);
    printf("Command: %s\n", appInfo->command);
    OVwFreeAppRegInfo(appInfo);
}
OVwSetRegContext(savedContext);
```

Implementation Specifics

OVwGetRegContext and OVwSetRegContext support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetRegContext or OVwSetRegContext, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See OVwGetApp in “OVwCreateApp(3)” on page 607.
- See “OVwGetFirstRegContext(3)” on page 717.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwGetSelections(3)

Purpose

Retrieves the current map selection list

Syntax

```
#include <OV/ovw.h>

OVwObjectIdList *OVwGetSelections(OVwMapInfo *map, char *actionId);
```

Description

OVwGetSelections returns a list of object IDs for those objects currently selected on the specified map. By supplying the name of a registered application action, OVwGetSelections will return the list of map selections only if they are all valid for the action.

Parameters

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or can be saved from the ovwMapOpen event using OVwCopyMapInfo.

actionId Specifies a pointer to the name of the action registered in the application's registration file whose selection rule should test whether the currently selected list of objects is valid. If actionId is NULL, OVwGetSelections returns all selected objects. Otherwise, a selection list is returned only if every selected object is valid according to the action's selection rule.

Return Values

If successful, OVwGetSelections returns a non-NULL OVwObjectIdList pointer. If unsuccessful, it returns a NULL pointer.

Error Codes

OVwGetSelections sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_FOUND] The named action has not been registered for this application.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_SUCCESS] Either there were no objects selected on the map or the selections were not valid for the specified action.

Examples

You can highlight the current list of selected objects by entering:

```
            OVwObjectIdList *objs;

            /* Get all objects currently selected */
            objs = OVwGetSelections(map, NULL);

            /* Highlight the selections */
            OVwHighlightObjects (map, objs, FALSE);
/* Free ID list memory */
OVwDbFreeObjectIdList(objs);
```

Implementation Specifics

OVwGetSelections supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetSelections, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.
- See OVwDbFreeObjectIdList in “OVwDbListObjectsByFieldValue(3)” on page 667.

OVwGetSubmapInfo(3)

Purpose

Returns submap information

Related Functions

OVwFreeSubmapInfo

Syntax

```
#include <OV/ovw.h>

OVwSubmapInfo *OVwGetSubmapInfo(OVwMapInfo *map, OVwSubmapId submapId);

void OVwFreeSubmapInfo(OVwSubmapInfo *submap);
```

Description

OVwGetSubmapInfo returns information about a submap on the open map. If the parent_object_id field of the OVwSubmapInfo structure is ovwNullObjectId, the submap is an orphan submap and has no parent object; otherwise, parent_object_id specifies the parent object of the submap.

OVwFreeSubmapInfo frees memory allocated for an OVwSubmapInfo structure. It should be used to free the OVwSubmapInfo structure returned by OVwGetSubmapInfo.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
submap	Specifies a pointer to the OVwSubmapInfo structure to free
submapId	Specifies the submap ID of the submap

Return Values

If successful, OVwGetSubmapInfo returns a pointer to an OVwSubmapInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwGetSubmapInfo sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap identified by submapId does not exist on the open map.

Implementation Specifics

OVwGetSubmapInfo and OVwFreeSubmapInfo support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetSubmapInfo or OVwFreeSubmapInfo, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwGetSymbolInfo(3)

Purpose

Returns symbol information

Related Functions

OVwFreeSymbolInfo

Syntax

```
#include <OV/ovw.h>

OVwSymbolInfo *OVwGetSymbolInfo(OVwMapInfo *map,
       OVwSymbolId symbolId);

void OVwFreeSymbolInfo(OVwSymbolInfo *symbol);
```

Description

OVwGetSymbolInfo returns information about a symbol on the open map.

OVwFreeSymbolInfo frees memory allocated for an OVwSymbolInfo structure. It should be used to free the OVwSymbolInfo structure returned by OVwGetSymbolInfo.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
symbol	Specifies a pointer to the OVwSymbolInfo structure to be freed.
symbolId	Specifies the symbol ID of the symbol.

Return Values

If successful, OVwGetSymbolInfo returns a pointer to an OVwSymbolInfo structure. If unsuccessful, it returns NULL.

Error Codes

OVwGetSymbolInfo sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwGetSymbolInfo and OVwFreeSymbolInfo support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwGetSymbolInfo or OVwFreeSymbolInfo, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwGetSymbolsByObject(3)

Purpose

Returns symbols for an object

Syntax

```
#include <OV/ovw.h>

OVwSymbolList *OVwGetSymbolsByObject(OVwMapInfo *map,
       OVwObjectId objectId);
```

Description

OVwGetSymbolsByObject returns a list of all the symbols that represent an object on the open map.

OVwFreeSymbolList, described in “OVwListSymbols(3)” on page 750, should be used to free the OVwSymbolList structure returned by OVwGetSymbolsByObject.

Generally, an object exists on a map when it is represented by a symbol on that map. Therefore, OVwGetSymbolsByObject will normally return at least one symbol. However, there are two cases in which an object can exist on a map without having an associated symbol:

- A submap is created with a parent object that is not yet represented by a symbol on the map.
- The Cut operation is used to cut the last symbol of an object to the clipboard, and the clipboard has not yet been cleared by another operation.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
objectId	Specifies the object ID of the object.

Return Values

If successful, OVwGetSymbolsByObject returns a pointer to an OVwSymbolList structure. If unsuccessful, it returns NULL.

Error Codes

OVwGetSymbolsByObject sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OBJECT_NOT_ON_MAP]	The object specified by objectId does not exist on the open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Libraries

When compiling a program that uses OVwGetSymbolsByObject, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwListSymbols(3)” on page 750.
- See “OVwApiIntro(5)” on page 560.
- See OVwFreeSymbolList in “OVwListSymbols(3)” on page 750.

OVwHighlightObject(3)

Purpose

Highlights objects on the map

Related Functions

OVwHighlightObjects

Syntax

```
#include <OV/ovw.h>

int OVwHighlightObject(OVwMapInfo *map, OVwObjectId object,
       OVwBoolean clearPrevious);

int OVwHighlightObjects(OVwMapInfo *map, OVwObjectIdList *objectList,
       OVwBoolean clearPrevious);
```

Description

OVwHighlightObject highlights the specified object in the open map, optionally clearing previously highlighted objects on the map. When OVwHighlightObject successfully highlights the specified object, the graphical interface opens a submap where there is a highlighted symbol for the object.

OVwHighlightObjects highlights a list of objects in the open map, optionally clearing previously highlighted objects on the map. If only one object is specified in the list of objects, and that object is successfully highlighted, the graphical interface opens a submap that has a highlighted symbol for that object.

Note: When multiple objects are specified in a call to OVwHighlightObjects, it is possible that some objects are not on the map; therefore, they are not highlighted. OVwHighlightObjects will return success in this case, but the return value of OVwError will be [OVw_OBJECT_NOT_ON_MAP]. Use OVwHighlightObject to ensure that every object is highlighted successfully.

Parameters

clearPrevious	Specifies a boolean flag, which, when TRUE, causes the graphical interface to clear currently highlighted objects on the map before highlighting the objects specified in the call. If it is FALSE, currently highlighted objects remain highlighted. The objects specified in the call are highlighted in addition to those objects already highlighted on the map.
map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
object	Specifies the object ID of the object to be highlighted.
objectList	Specifies a pointer to a list of object IDs of the objects to be highlighted.

Return Values

If successful, OVwHighlightObject and OVwHighlightObjects return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwHighlightObject and OVwHighlightObjects set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OBJECT_NOT_ON_MAP]	The object does not exist on the open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following example shows how an application can highlight the current list of selected objects:

```
OVwObjectIdList *objs;

/* Get the current selection list */
objs  = OVwGetSelections(map, NULL);

/* Highlight the  selections */
OVwHighlightObjects (map, objs, FALSE);
/* Free list of IDs */
OVwDbFreeObjectIdList(objs);
```

Implementation Specifics

OVwHighlightObject and OVwHighlightObjects support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwHighlightObject or OVwHighlightObjects, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See OVwDbFreeObjectIdList in “OVwDbListObjectsByFieldValue(3)” on page 667.

OVwInit(3)

Purpose

Initializes an application's connection to the NetView for AIX program.

Syntax

```
#include <OV/ovw.h>

int OVwInit();
```

Description

OVwInit initializes internal API data structures and the communications channel between an NetView for AIX application and the NetView for AIX program. It must be called before any other EUI API call. Each application or process can call OVwInit only once; subsequent calls will result in errors. If your application calls OVwInit, then before it exits it should call OVwDone.

Return Values

If successful, OVwInit returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwInit sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ALREADY_INITIALIZED]	The API has been initialized with a prior call to OVwInit.
[OVw_CONNECT_ERROR]	A failure occurred when attempting to connect to the NetView for AIX program.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OVW_NOT_RUNNING]	The application was not invoked from the NetView for AIX program.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

Examples

The following code fragment shows how to initialize the EUI API prior to other OVw calls:

```
if (OVwInit() < 0) {
    fprintf(stderr, "foo: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}

OVwAddCallback(ovwEndSession, NULL, (OVwCallbackProc)endCB, NULL);
```

Libraries

When compiling a program that uses OVwInit, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwDone(3)” on page 685.
- See “OVwError(3)” on page 688.
- See “OVwApiIntro(5)” on page 560.

OVwIsIdNull(3)

Purpose

Tests and compares EUI API IDs

Related Functions

OVwIsIdEqual

Syntax

```
#include <OV/ovw.h>

OVwBoolean OVwIsIdNull(id)

OVwBoolean OVwIsIdEqual(id1, id2)
```

Description

OVwIsIdNull and OVwIsIdEqual are macros for testing and comparing IDs used in the EUI API. These macros should be used with object IDs (OVwObjectId), field IDs (OVwFieldId), submap IDs (OVwSubmapId), and symbol IDs (OVwSymbolId). These macros are defined in the <OV/ovw_types.h> header file, which is included by the <OV/ovw.h> header file.

OVwIsIdNull is a macro that returns TRUE if id has a null value; otherwise, it returns FALSE.

OVwIsIdEqual is a macro that returns TRUE if id1 and id2 are equal; otherwise, it returns FALSE. Both IDs should be of the same type.

Parameters

id Specifies an object ID, field ID, submap ID, or symbol ID.

id1 Specifies an object ID, field ID, submap ID, or symbol ID.

id2 Specifies an object ID, field ID, submap ID, or symbol ID.

Warning

Failure to use OVwIsIdNull and OVwIsIdEqual may result in future compatibility problems if the implementation of IDs in the EUI API is changed.

Libraries

When compiling a program that uses OVwIsIdNull or OVwIsIdEqual, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See “OVwApiIntro(5)” on page 560.

OVwListObjectsOnMap(3)

Purpose

Lists objects on a map

Related Functions

OVwFreeObjectList

Syntax

```
#include <OV/ovw.h>

OVwObjectList *OVwListObjectsOnMap(OVwMapInfo *map,
       OVwFieldBindList *capabilitySet);

void OVwFreeObjectList(OVwObjectList *objectList);
```

Description

OVwListObjectsOnMap returns a list of objects on the open map.

The optional parameter fieldValues enables filtering to be done based on the values of fields specified in the list. Capability field values can be specified to get a list of different kinds of objects. The filter is a logical AND of fields in the argument fieldValues. If fieldValues is NULL, all objects on the open map are returned.

OVwFreeObjectList frees the memory allocated for an OVwObjectList structure. It should be used to free the OVwObjectList structure returned by OVwListObjectsOnMap.

Parameters

fieldValues	Specifies a pointer to an optional filter based on a list of field values.
map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
objectList	Specifies a pointer to an OVwObjectList structure to be freed.

Return Values

If successful, OVwListObjectsOnMap returns a pointer to an OVwObjectList structure. If unsuccessful, it returns NULL. The number of items in the object list may be zero if no object matches the filter.

Error Codes

OVwListObjectsOnMap sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.

[OVw_FIELD_NOT_FOUND]	A field ID in the fieldValues argument does not indicate a field in the database.
[OVw_FIELD_TYPE_MISMATCH]	The field data type provided in an OVwFieldBinding structure in fieldValues does not match the field data type defined for the given field ID.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwListObjectsOnMap and OVwFreeObjectList support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwListObjectsOnMap or OVwFreeObjectList, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwListSubmaps(3)

Purpose

Lists submaps on a map

Related Functions

OVwFreeSubmapList

Syntax

```
#include <OV/ovw.h>

OVwSubmapList *OVwListSubmaps(OVwMapInfo *map, char *appName,
       int submapType, OVwFieldBindList *parentFieldValues);

void OVwFreeSubmapList(OVwSubmapList *submapList);
```

Description

OVwListSubmaps returns a filtered list of submaps from the open map. A logical AND of the three filters (appName, submapType, and parentFieldValues) determines which submaps are returned. If a NULL value is specified for all of these filters, all submaps on the map are returned.

OVwFreeSubmapList frees the memory allocated for an OVwSubmapList structure. OVwFreeSubmapList should be called to free the OVwSubmapList structure returned by OVwListSubmaps.

Parameters

appName — Specifies a pointer to the name of the application that created the submaps. An appName NULL value matches any application. OVwGetAppName returns the name of the application making the call.

map — Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

parentFieldValues — Specifies a pointer to a filter, based on a list of field values that are compared with the field values of the parent objects of submaps. A logical AND of the fields in the list is used. If parentFieldValues is NULL, no filtering is done on the field values of the parent object.

submapList — Specifies a pointer to an OVwSubmapList structure to free.

submapType — Specifies a filter for the submap type set by the application that created the submap. The value of submapType is application-specific and only unique within the scope of the creating application. A submapType value of ovwAnySubmapType matches any submap.

Return Values

If successful, OVwListSubmaps returns a pointer to an OVwSubmapList structure. If unsuccessful, it returns NULL. The number of items in the submap list may be zero if no submap matches the filters.

Error Codes

OVwListSubmaps sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	The application specified by appName is not registered in an application registration file.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_DB_CONNECTION_LOST]	The connection to ovwdb was lost.
[OVw_FIELD_NOT_FOUND]	A field ID in the fieldValues argument does not indicate a field in the database.
[OVw_FIELD_TYPE_MISMATCH]	The field data type provided in an OVwFieldBinding structure in fieldValues does not match the field data type defined for the given field ID.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

You can return all submaps on the open map by entering the following code:

```
OVwListSubmaps(map, NULL, ovwAnySubmapType, NULL);
```

You can return all submaps created by the calling application by entering the following code:

```
char *appname = OVwGetAppName();

OVwListSubmaps(map, appname, ovwAnySubmapType, NULL);
```

You can return all submaps of type 2 created by the IP Map application by entering the following code:

```
OVwListSubmaps(map, "IP Map", 2, NULL);
```

The following example shows how to print the names of all submaps on the open map:

```
        int i;
        OVwSubmapList *submap_list;
        OVwMapInfo *map = OVwGetMapInfo();

        submap_list = OVwListSubmaps(map, NULL, ovwAnySubmapType, NULL);
        for (i = 0; i < submap_list->count; i++) {
                printf("%s\n", submap_list→submaps[i].submap_name);
        }
        OVwFreeSubmapList(submap_list);
OVwFreeMapInfo(map);
```

Implementation Specifics

OVwListSubmaps and OVwFreeSubmapList support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwListSubmaps or OVwFreeSubmapList, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See “OVwError(3)” on page 688.
- See “OVwGetAppName(3)” on page 701.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwListSymbols(3)

Purpose

Lists symbols on a submap

Related Functions

OVwFreeSymbolList

Syntax

```
#include <OV/ovw.h>

OVwSymbolList *OVwListSymbols(OVwMapInfo *map, OVwSubmapId submapId,
       OVwPlaneType plane, char *appName);

void OVwFreeSymbolList(OVwSymbolList *symbolList);
```

Description

OVwListSymbols returns a filtered list of symbols on a submap of the open map. A logical AND of the appName and plane filters determines which symbols are returned. If appName has a NULL value and plane has the value ovwAllPlanes, all symbols on the submap are returned.

The appName parameter provides a filter for the set of symbols in a submap in which a particular application is interested. The apps field of the OVwSymbolInfo structure lists the applications that have expressed an interest in the symbol. This list is initialized with the application creating the symbol. OVwSetSymbolApp and OVwClearSymbolApp can be used to modify this list. This mechanism enables a given application to define a set of symbols, on a particular submap, that conforms to the semantics of that submap as defined by the application.

A symbol appears on the application plane of the submap only if there is at least one application that is interested in it. If no application is interested in the symbol, it appears on the user plane.

OVwFreeSymbolList frees the memory allocated for an OVwSymbolList structure. Use OVwFreeSymbolList to free the OVwSymbolList structure, which is returned by OVwListSymbols.

Parameters

appName Specifies a pointer to the name of the application whose symbols should be returned. OVwGetAppName returns the name of the calling application. A NULL value matches any application.

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

plane Specifies a filter for the plane on which the symbols exist. The permitted values are defined in the <OV/ovw.h> header file:

ovwAllPlanes	Return symbols on all planes.
ovwAppPlane	Return only symbols on the application plane.
ovwUserPlane	Return only symbols on the user plane.
	If zero (0) is specified, no symbols will be returned.

submapId Specifies the ID of the submap.

symbolList Specifies the OVwSymbolList structure to be freed.

Return Values

If successful, OVwListSymbols returns a pointer to an OVwSymbolList structure. If unsuccessful, it returns NULL. The number of items in the symbol list might be zero if no symbol matches the filters.

Error Codes

OVwListSymbols sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	The application specified by appName is not registered in an application registration file.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap specified by submapId does not exist on the open map.
[OVw_SUBMAP_PLANE_INVALID]	The argument plane has a bit set that is not valid.

Examples

You can return all symbols on the specified submap by entering the following code:

```
OVwListSymbols(map, submap_id, ovwAllPlanes, NULL);
```

You can return all symbols of the calling application by entering the following code:

```
char *appname = OVwGetAppName();
OVwListSymbols(map, submap_id, ovwAppPlane, appname);
```

Implementation Specifics

OVwListSymbols and OVwFreeSymbolList support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwListSymbols or OVwFreeSymbolList, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetAppName(3)” on page 701.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwSetSymbolApp(3)” on page 808.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwListSymbolTypes(3)

Purpose

Returns symbol type information

Related Functions

OVwListSymbolTypeCaps
OVwFreeSymbolTypeList

Syntax

```
#include <OV_ovw.h>

OVwSymbolTypeList *OVwListSymbolTypes();

OVwFieldBindList *OVwListSymbolTypeCaps(OVwSymbolType symbol);

void OVwFreeSymbolTypeList(OVwSymbolTypeList *symbolTypeList)
```

Description

OVwListSymbolTypes is used to return a list of all the currently registered symbol types. These symbol types are used primarily to present a graphic representation of an object. They are secondarily used to determine initial capability-field values of objects.

OVwListSymbolTypeCaps is used to return a list of the capability-field values which would be set if the user added a map object using this symbol type. It is also possible for an application to use the symbol type to define an initial set of capability-field values for an object. See “OVwCreateSymbol(3)” on page 623 for a more complete description of this latter use of symbol capabilities.

OVwFreeSymbolTypeList frees the memory allocated for an OVwSymbolTypeList structure. OVwFreeSymbolTypeList should be used to free the OVwSymbolTypeList structure returned by OVwListSymbolTypes.

OVwDbFreeFieldBindList should be used to free the OVwFieldBindList structure returned by OVwListSymbolTypeCaps.

Parameters

symbol Specifies the symbol type for which to return the capability list.

Return Values

If successful, OVwListSymbolTypes and OVwListSymbolTypeCaps return pointers to the requested lists. If unsuccessful, they return NULL.

Error Codes

OVwListSymbolTypes and OVwListSymbolTypeCaps set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwListSymbolTypes and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwListSymbolTypes or its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See OVwDbFreeFieldBindList in “OVwDbGetFieldValues(3)” on page 654.

OVwLockRegUpdates(3)

Purpose

Acquires permission to modify registration information

Related Functions

OVwUnlockRegUpdates

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwLockRegUpdates(OVwBoolean block);

int OVwUnlockRegUpdates();
```

Description

OVwLockRegUpdates acquires permission for the application to make subsequent calls that modify NetView for AIX registration information. Such calls include OVwActionRegistration, OVwAddMenuItem, OVwAddMenuItemFunction, OVwAppRegistration, OVwMenuRegistration, and OVwMenuItemRegistration. Only one NetView for AIX application is permitted to modify registration information at a time, so the lock is needed to enforce this mutual exclusion.

OVwUnlockRegUpdates releases previously acquired update permissions.

Parameters

block If TRUE, OVwLockRegUpdates will not return until the lock is acquired. If FALSE, OVwLockRegUpdates will return immediately with an indication of success or failure.

Return Values

If successful, OVwLockRegUpdates and OVwUnlockRegUpdates return 0 (zero). If unsuccessful, they return –1.

Error Codes

OVwLockRegUpdates and OVwUnlockRegUpdates set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED] The registration update permissions could not be acquired.

Libraries

When compiling a program that uses OVwLockRegUpdates or OVwUnlockRegUpdates, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwMainLoop(3)

Purpose

Defines NetView for AIX graphical user interface MainLoop, which continuously processes NetView for AIX events

Syntax

```
#include <OV/ovw.h>

void OVwMainLoop();
```

Description

OVwMainLoop is a macro that defines a while loop that continuously processes NetView for AIX events and application-registered input events. Applications must either call OVwMainLoop or OVwXtMainLoop for application callbacks, registered with OVwAddCallback, to be functional.

OVwMainLoop returns only if the connection to the NetView for AIX program is closed, and there are no more active input sources previously registered with OVwAddInput. Applications should exit in response to the ovwEndSession event rather than depending on OVwMainLoop to return.

Examples

The following code fragment illustrates how a minimal NetView for AIX application uses OVwMainLoop:

```
void
EndSessionProc(void *userData, OVwEventType type,
        OVwBoolean normalEnd)
{
        if (normalEnd)
                printf("ovw terminated normally.\n");
        else
                printf("ovw terminated abnormally.\n");
        OVwDone();
        exit(!normalEnd);
}

main()
{
        if (OVwInit() < 0) {
            fprintf(stderr, "example: %s\n", OVwErrorMsg(OVwError()));
            exit(1);
        }

        OVwAddCallback(ovwEndSession, NULL,
                        (OVwCallbackProc)EndSessionProc, NULL);

        OVwMainLoop();
}
```

Libraries

When compiling a program that uses OVwMainLoop, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwAddInput(3)” on page 543.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwMapCloseCB(3)

Purpose

Functions as a callback for a map-close event

Syntax

```
#include <OV/ovw.h>

void (*OVwMapCloseCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, time_t closing_time);
```

Description

To receive an event indicating that a map is being closed, use OVwAddCallback to register a callback function of type OVwMapCloseCB to be called when an ovwMapClose event is generated. A map close event is generated when a map is closed through the graphical interface. A map close event implies that the submaps within the map were closed. A submap close callback will not be generated when a map close event occurs.

All applications receiving an ovwMapClose event must call OVwAckMapClose to acknowledge the close of the map. The map will be closed only when all applications receiving the ovwMapClose event have called OVwAckMapClose. This is done so that all applications are in agreement when a map is closed and applications do not mistakenly perform operations on a map that has already been closed.

The closing_time parameter is a proposed closing time for the map, based on the time the map-close event was generated. When calling OVwAckMapClose, an application can agree with the default closing time by returning the value of closing_time or a default value of 0 (zero). Alternately, an application that is interrupted in the middle of making map updates can indicate an earlier map closing time to indicate that there are still updates that need to be performed on the map. The actual closing time for the map will be the earliest time indicated by any application. This final closing time will be available in the last_closed_time field of the OVwMapInfo structure passed in the ovwMapOpen event when the map is next opened.

Because application acknowledgement is required to close a map, response time for closing a map might be slow if applications are busy doing other processing. In order to avoid this, it is strongly recommended that applications receiving the ovwMapClose event use the OVwPeekOVwEvent routine to regularly check for the ovwMapClose event, especially during lengthy processing. If OVwPeekOVwEvent returns TRUE for an ovwMapClose event, the application should ignore subsequent events or process them as quickly as possible until the map close event is received. Also, once it is discovered through OVwPeekOVwEvent that a map close event is imminent, the application can begin determining what map close time to use.

There is a map close time-out, so that if all applications do not respond within the number of seconds specified by the NetView for AIX X resource closeTimeout, the map is closed anyway. The default is **two minutes**.

Parameters

closing_time	Returns the proposed closing time for the map. If acceptable, this time can be passed as the close_time parameter of OVwAckMapClose.
map	Specifies a pointer to the OVwMapInfo structure for the map that is being closed.

type	Specifies the type of event that caused this callback to be invoked, namely ovwMapClose.
userData	Specifies user data provided when the callback was added.

Examples

The following code fragment illustrates how to register a callback routine that receives a map-close event:

```
void
mapCloseProc(void *userData, OVwEventType type,
        OVwMapInfo *map, time_t closing_time)
{
        time_t new_close_time = (time_t) 0;

        /* do cleanup for map being closed */

        /*
         * If necessary, compute an earlier new_close_time
         * based on updates that need to be done next time
         * the map is opened.
         */
        OVwAckMapClose(map, new_close_time);
}

OVwAddCallback(ovwMapClose, NULL,
        (OVwCallbackProc) mapCloseProc, NULL);
```

Implementation Specifics

OVwMapCloseCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwMapCloseCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAckMapClose(3)” on page 528.
- See “OVwAddCallback(3)” on page 539.
- See “OVwMapOpenCB(3)” on page 761.
- See “OVwPeekOVwEvent(3)” on page 763.
- See “OVwApiIntro(5)” on page 560.

OVwMapOpenCB(3)

Purpose

Functions as a callback for a map open event

Syntax

```
#include <OV/ovw.h>

void (*OVwMapOpenCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwFieldBindList *configParams);
```

Description

To receive an event indicating that a new map has been opened, use OVwAddCallback to register a callback function of type OVwMapOpenCB to be called when an ovwMapOpen event is generated.

Note: The ovwMapOpen event initially occurs at startup time, before registered applications are have been started. Applications do not receive a callback for this initial map open event.

The permissions field of the OVwMapInfo structure returned will indicate whether the map has been opened ovwReadOnly or ovwReadWrite. If a map is opened ovwReadOnly, all calls that modify the open map will result in an OVw_MAP_READ_ONLY error, except for calls to change status. See “OVwSetStatusOnObject(3)” on page 803 for more information.

The configParams parameter points to the current values of any map-specific application configuration fields enrolled by the application through an application registration file. If no fields were enrolled, the value of this parameter will be NULL.

If the application is enabled for the open map, as determined by the application's configuration parameters, the application should begin the process of synchronizing the map with the latest status and topology information. This initial update should be enclosed between OVwBeginMapSync and OVwEndMapSync calls.

The last_closed_time field of the OVwMapInfo structure gives the last time a read-write version of the map was closed. This time can be used to determine what changes are needed to update the map. A map that is being opened for the first time since it was created will have a last_closed_time of 0 (zero).

Parameters

configParams	Specifies a pointer to the application configuration field values for the open map. This parameter will be NULL if the application has not enrolled any application configuration fields through an application registration file.
map	Specifies a pointer to an OVwMapInfo structure providing information about the map just opened. This parameter can be saved with OVwCopyMapInfo (see “OVwGetMapInfo(3)” on page 719) for later use as the map parameter in routines that operate on the open map.
type	Specifies the type of event that caused this callback to be invoked, namely ovwMapOpen.
userData	Specifies user data provided when the callback was added.

Examples

The following code fragment illustrates how to register a callback routine that receives map-open events:

```
void
mapOpenProc(void *userData, OVwEventType type,
       OVwMapInfo *map, OVwFieldBindList *configParams)
{
       /* check configParams */
       if (enabled_for_map) {
              OVwBeginMapSync(map);
              /*
               * Update map with status changes since
               * map->last_closed_time.
               */
              if (map->permissions == ovwMapReadWrite)  {
                     /* update map with topology changes */
               }
               OVwEndMapSync(map);
        }
}

OVwAddCallback(ovwMapOpen, NULL,
        (OVwCallbackProc) mapOpenProc, NULL);
```

Implementation Specifics

OVwMapOpenCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwMapOpenCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See “OVwAddCallback(3)” on page 539.
- See “OVwBeginMapSync(3)” on page 573.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwSetStatusOnObject(3)” on page 803.
- See “OVwMapCloseCB(3)” on page 759.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwPeekOVwEvent(3)

Purpose

Checks for specific NetView for AIX or input events

Related Functions

OVwPeekInputEvent

Syntax

```
#include <OV/ovw.h>

OVwBoolean OVwPeekOVwEvent(OVwEventType event);

OVwBoolean OVwPeekInputEvent(OVwInputId id);
```

Description

OVwPeekOVwEvent enables your application to determine whether a particular type of NetView for AIX event is awaiting processing. This is particularly helpful in applications that must handle the ovwMapClose event promptly. By checking for ovwMapClose at certain intervals during lengthy processing, you can quickly stop processing so that the map close can be received and acknowledged promptly. See "OVwAckMapClose(3)" on page 528 and "OVwMapCloseCB(3)" on page 759 for more details on handling the ovwMapClose event.

OVwPeekInputEvent provides a similar mechanism for application-registered input sources by enabling your application to determine whether a registered input source has input waiting to be processed.

Parameters

event Specifies an NetView for AIX event as defined in the <OV/ovw.h> header file

id Specifies an OVwInputId, which results from a prior call to OVwAddInput

Return Values

If there are pending NetView for AIX events, OVwPeekOVwEvent returns TRUE. If there are no pending NetView for AIX events, OVwPeekOVwEvent returns FALSE.

If there are any pending input events, OVwPeekInputEvent returns TRUE. If there are no pending NetView for AIX events, OVwPeekInputEvent returns FALSE.

If a failure occurs, the functions return FALSE.

Error Codes

OVwPeekOVwEvent and OVwPeekInputEvent set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following code fragment illustrates how an application might use OVwPeekOVwEvent to stop map synchronization if the map has been closed:

```
OVwBeginMapSync(map);
syncing = TRUE;
mapClosing = FALSE;

while (syncing) {
        if (OVwPeekOVwEvent(ovwMapClose) == TRUE) {
                syncing = FALSE;
                mapClosing = TRUE;
        } else {
                /* Process a few map synchronization steps */
        }
}

OVwEndMapSync(map);
```

Libraries

When compiling a program that uses OVwPeekOVwEvent or OVwPeekInputEvent, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwPending(3)” on page 765.
- See “OVwApiIntro(5)” on page 560.

OVwPending(3)

Purpose

Tests for pending NetView for AIX or application-registered events

Syntax

```
#include <OV/ovw.h>

OVwBoolean OVwPending();
```

Description

OVwPending returns a boolean value indicating whether there is an NetView for AIX event or application-registered input event waiting to be processed.

Return Values

If there are pending events, OVwPending returns TRUE. If there are no pending events, it returns FALSE. If OVwPending experiences a failure, it returns FALSE.

Error Codes

OVwPending sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX process was lost.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

A fragment from the code for OVwXtMainLoop demonstrates the use of OVwPending:

```
while (1)  {
    if (OVwPending())
        OVwProcessEvent();
    else
        XtProcessEvent(XtIMAll);
}
```

Libraries

When compiling a program that uses OVwPending, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwProcessEvent(3)” on page 767.
- See “OVwApiIntro(5)” on page 560.

OVwProcessEvent(3)

Purpose

Processes a pending NetView for AIX event

Syntax

```
#include <OV/ovw.h>

int OVwProcessEvent();
```

Description

OVwProcessEvent processes pending NetView for AIX events or application-defined input events. Callbacks are invoked for these events based on previous registration with OVwAddCallback, OVwAddActionCallback, or OVwAddInput.

Return Values

If successful, OVwProcessEvent returns 0 (zero). If unsuccessful, it returns –1.

Error Codes

OVwProcessEvent sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred..
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

The following fragment shows how OVwMainLoop uses OVwProcessEvent:

```
while(1) {
    if (OVwProcessEvent() < 0)
        break;
}
```

Libraries

When compiling a program that uses OVwProcessEvent, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwPending(3)” on page 765.
- See “OVwApiIntro(5)” on page 560.

OVwRegIntro(5)

Purpose

Provides an overview of NetView for AIX graphical user interface registration files

Description

NetView for AIX graphical user interface registration files are a mechanism for:

- Integrating applications into the NetView for AIX graphical user interface
- Defining symbol types for the NetView for AIX graphical user interface map
- Creating fields in the NetView for AIX graphical user interface Object Database (see ovwdb(8))
- Enabling managers to access your agent or application through the NetView for AIX program

When the NetView for AIX program processes a set of registration files, it creates a list of all the files in a registration directory, collates the list, then parses and processes each of the files in this collated order.

Use the NetView for AIX program's -verify option to ensure the accuracy of the registration files before starting an NetView for AIX session after registration files have been modified.

Application Integration

Application registration files define how an application integrates with NetView for AIX graphical user interface. This information includes details, such as:

- The definition of specific actions the application can perform
- A command line to start the application
- Flags telling the NetView for AIX program how the application command should be started and managed
- Graphical interface menus and menu items that start or signal the application to perform defined actions
- Descriptions of dialog boxes the application will use for map semantic information

Applications that are started from the NetView for AIX program and those that use the EUI API must be registered through an application registration file. The file provides the necessary information for the NetView for AIX program to invoke and manage the application processes.

An application can have certain actions defined for it. These, too, are specified in the registration file and can be linked to menu items, within the registration file, or to executable symbols, through the EUI API or the graphical user interface.

Some applications manage semantics, or semantic planes, of graphical interface maps. These applications need to specify how certain dialog boxes look for various semantic operations. Within the registration file, the application enrolls fields for these dialog boxes, based on rules about the objects it will manage. Details on how to use field enrollment in connection with the EUI API are available in the *NetView for AIX Programmer's Guide*.

Application registration files are located in the /usr/OV/registration/$LANG directory. The NetView for AIX application parses the files in collated, file name order. Once all the application registration files have been parsed, application menus are registered in the menu bar. The NetView for AIX graphical user

interface menu registration is processed first, followed by all other applications in application name order. That is, application Alpha's menu items are registered before application Omega's.

Application Block

The Application block contains information about how an application is integrated with the NetView for AIX program. Only one application can be registered in a single application registration file. See Example of Application Registration on page 791 for an example registration file.

The application name must uniquely identify an application. For example, each of the following sample application registrations attempt to define the same application, Sample App. However, both application registrations cannot be registered.

First sample application registration

```
Application "Sample App" {
        Command "/usr/local/bin/checknode -test";
                :
                :
}
```

Second sample application registration

```
Application "Sample App" {
        Command "/usr/bin/foo -x -y -z ${OVwSelections}";
                :
}
```

The NetView for AIX application emits a warning if there are multiple registrations for an application and ignores duplicate registrations.

Command and Process Flags

The Command statement contains process flags and a command string that has command-line arguments used to invoke the application. This declaration outside an action declaration is global, in that any actions defined later assume that the application process is started with this command. This command can be overridden within an action declaration. This global declaration can be omitted, provided there are command statements within all action declarations.

The NetView for AIX application executes the command string by executing the /bin/sh -c exec command. For the command invocation to succeed, the first element of the command string must be the path of an executable file. The full syntax of sh(1) can be used in the command string, including references to environment variables inherited from the NetView for AIX program. Additionally, the NetView for AIX program sets the following environment variables, which the application inherits:

OVwSelections
: This is a list, separated by blanks, of the selection names of the objects in the selection list at the time the application is invoked. When an object menu is in use, OVwSelections is set to the selection name of the object from which the action was selected.

OVwNumSelections
: This is set to the number of objects in the selection list at the time the application is invoked. When an object menu is in use, OVwNumSelections is set to one.

*OVwSelection**n** (**n**=1, 2, ..., 10)*
: Each of these environment variables contains the selection name of an object in the selection list at the time the application is invoked. OVwSelection1 is set to the name of the first object selected, OVwSelection2 the second, and so on, up to OVwSelection10 which is set to the selection name of the tenth object selected. The ordering of the selected objects for these vari-

ables is the order in which they were selected on the map. When an object menu is in use, OVwSelection*n* contains the selection name of the object menu from which the action was selected.

OVwActionID
This is the name of the action declaration block, as defined in the registration file, by which the application was invoked.

OVwMenuItem
If an application action is invoked from a menu, this variable will be set and will contain the label of the menu item that caused the action.

The process flags listed in the command statement tell the NetView for AIX program how to manage the application process or processes. They are specified with the Command in the application block, as shown in the following example:

```
Application "Sample App" {
        Command -Shared -Initial -Restart "/usr/bin/foo -x -y -z
        ${OVwSelections}";
                :
}
```

Valid process flags are:

-Initial
This flag tells the NetView for AIX program to start the application when the NetView for AIX program starts. The NetView for AIX program will invoke the application with the command specified by the Command line in the application block. By default, this flag is not enabled and an application is only started when an application action is triggered by a menu item or an executable symbol.

-Shared
This flag tells the NetView for AIX program that the application process instance is shared across actions. Once the process is started by a menu item, executable symbol, or a set Initial flag, it continues to run until the command instance exits. The process is then notified by the EUI API when actions are requested by menu items or executable symbols. Applications that do not use the EUI API can have the Shared flag enabled, meaning only that the first instance of the application will be the only instance running until it exits. It must use the EUI API to intercept further action requests from menu items or executable symbols.

-Restart
This flag tells the NetView for AIX program that the application is a required application for normal operation and should be restarted if it exits. This flag is intended for use by applications that manage semantics of graphical interface maps and should always be present for the duration of an NetView for AIX session.

Application Description

The Description block provides a brief description of the application for use in the Help..Indexes..Applications dialog box, which displays all the applications installed and registered with the NetView for AIX program.

The description block contains a comma-separated list of strings. Within the application index, each string will appear on a separate line.

Application Copyright

The Copyright block provides the copyright string used in “Help..Indexes..Applications” dialog box. Like the description, this block contains a comma-separated list of strings. Within the application index, each string appears on a separate line.

Application Version

The version statement defines a string which is the application version information. This version string appears in the Help..On Version and Help..Indexes..Applications dialog boxes.

Help Directory

The HelpDirectory statement specifies the name of the directory where the application’s help files reside. These files are presented in response to help requests by the NetView for AIX graphical user interface help system, ovhelp. The directory is listed as a flat name, not a path, assuming that the directory is located in relation to /usr/OV/help/$LANG.

For example:

```
HelpDirectory "IPMap";
```

Name Field

The NameField section provides a mechanism for the application to quickly access the selected objects by some name field other than the selection name. The environment variable $OVwSelections contains, by default, the selection names of all the objects in the selection list at application startup. In some cases, that name can be different from the names that the application is constructed to handle. For example, the telnet command can be specified as a menu item. Generally, the selection name for objects on the map is the hostname; however, users can select object names. To indicate that the command gets only host names, the application registration can contain a NameField section indicating that objects should be named by host name.

For example:

```
Application "Telnet" {
      Command "/usr/bin/X11/xterm -e /usr/bin/telnet ${OVwSelection1}";
      NameField "IP Hostname";
              :
 }
```

The string `IP Hostname` refers to the object field by the same name. Any registered name field can be used in this binding. If an object that does not have a host name appears in the selection list, any menu items by which the application is invoked will be grayed out.

The NameField section accepts a list of field names. The first field that is defined for an object will be used in the selection list environment settings. In the following example, if an object that does not have an IP hostname were selected, its selection name would be used.

```
Application "Telnet" {
        Command "/usr/bin/telnet ${OVwSelection1}";
        NameField "IP Hostname", "Selection Name";
                :
}
```

Application Menu Integration

The MenuBar and Menu sections provide a specification of how the application interfaces with the graphical interface menu bar. Both are functionally equivalent, with one exception. The MenuBar block declares menu selections for top-level menus on the menu bar. The Menu block declares selections for menu cascades included in the top-level menus (top-level meaning the first cascade beneath a menu bar item). Thus, the scope of the MenuBar menuId is global (across applications) and also specifies the label

of the item on the menu bar. The scope of the Menu menuId is limited to the Application block and has no influence on the label of the button that brings up the menu cascade.

The Menu and MenuBar sections contain declarations of menu items for the application. Because the body of the Menu and MenuBar sections are the same, discussion of menu item declaration is deferred to the Menu section.

MenuBar Registration

The MenuBar section provides registration of new menu bar selections and registration of menu items within the top-level menus.

MenuName

The MenuName for the MenuBar section serves to distinguish MenuBar declarations and to name top-level (menu bar) menus. This MenuName is global in scope. It corresponds to the label for the menu bar. For example, the following code fragment is a declaration that describes entries for the application under the Administer menu on the menu bar.

```
MenuBar Administer _A {
        :
}
```

The Help MenuName is reserved to be the menubar item that is attached to the right side of the menubar.

Mnemonic

Along with the label for the menu bar item, an application can specify an optional mnemonic in the declaration of the menu. For example, an application that is registering for a new menu bar item Configure can specify the following to add a new top-level menu on the menu bar called Configure with the mnemonic character C.

```
MenuBar "Configure" _C {
        :
}
```

The MenuName Configure is now global in scope; other applications can register menus under this same top-level menu. If there are multiple declarations for a top-level menu that specify different mnemonics, the NetView for AIX program emits a warning and uses the first mnemonic registered for that cascade.

Precedence

The precedence is an integer value from 0 (zero) to 100 enclosed in a < and > symbol, which weights the importance of a menu bar cascade or item on the Tool window. Menu bar cascades or tools are listed in the menu bar or Tool window according to precedence and, within items of the same precedence, the order in which they are registered (collated order of application name). As shown in the following example, the NetView for AIX program registers the File cascade as the leftmost cascade by giving it the highest precedence value and by listing it first within the registration file:

```
MenuBar <100> "File" _F {
        :
}
```

Similarly, a tool called GraphDemo can be positioned at the top of the Tool window by giving it the highest precedence value:

```
Tool <100> "GraphDemo" {
        :
}
```

Menu Registration

The Menu section enables an application to specify a group of menu items to appear within a single menu cascade. For the declared menu to be useful, it must be associated with some menu item with the f.menu function.

The following example illustrates Menu registration:

```
Application “Mail Manager”
{

    MenuBar “Monitor”
    {
        “Mail”  f.menu “MailMenu”;
    }

    ObjectMenu
    {
        “Menuitem”  f.action “Menuitem”;
        “Menuitem”  f.action “Menuitem”;
    }

    Tool “Label”
    {
        Icon Bitmap “Filename”
        LabelColor “Color”;
        DragBitmap “Filename”;
        SelectionMechanism “drag-drop”,“double-click”;
        Action “Action Name”;
    }

    Menu “MailMenu”
    {
        “Mail Log”      f.action “MailLog”;
        “Mail Queue”    f.action “Mailq”;
    }

        :
        :
}
```

In the previous example, the application connects with the Monitor menu on the main menu bar. A menu item named Mail selects the MailMenu cascade menu that contains at least two items, Mail Queue and Mail Log.

MenuID

A MenuID provides a reference to the specified group of menu items. The sole purpose of the MenuID is to provide a way for the application to connect the group of menu items to a menu cascade button. The scope of this MenuID is limited to the application registration. An application can declare a group of menu items using the same MenuID that another application uses for another group of menu items. MailMenu is a MenuID in the preceding example.

MenuItem Registration

Menu items are declared within the Menu and MenuBar sections.

The following example is a sample MenuBar section.

```
MenuBar <100> File _F
{
    <100>   “New Map...”            _N      f.new_map;
    <100>   “Open Map...”           _O      f.avail_maps;
    <100>   “Describe Map...”       _M      f.map_desc;
    <100>   “Refresh Map”           _R      f.refresh_map;
    <100>   “Save Map As...”        _A      f.save_map;
    <100>   “Delete Map...”         _D      f.avail_maps;
    <100>   “Map Snapshot”          _h      f.menu “Map Snapshot”;
    <0>     “Exit”                  _E      Cntl<Key>E        f.exit;
```

A menu item is composed of the following components:

Precedence

The precedence value is optional. It can range from 0-to-100 enclosed by a < and > symbol and, by default, is set to **50**. This is an integer value that weights the importance of a menu item. Menu items are listed in graphical interface menus according to precedence and, within items of the same precedence, the order in which they were registered (the collated order of the application name).

Label

The label for the menu item as it should appear in a menu. It is a required field.

Mnemonic

A character that enables keyboard traversal of a menu. The mnemonic declaration begins with an underscore followed by the mnemonic character needed for the menu selection.

The mnemonic is an optional field. If another mnemonic is already specified, by another application, for the same menu item, the NetView for AIX program issues a warning and the previously defined mnemonic is used instead.

Accelerator

A key sequence that invokes a menu selection without displaying the menu.

This is an optional field. If the menu item already has an accelerator associated with it, through another application, the NetView for AIX program issues a warning and uses the previously-defined accelerator.

Function

A function describes the behavior of each menu selection. Functions begin with the two characters, f., followed by a name. A special function named “!” provides easy integration of shell commands with graphical interface menus.

“!” “command line”
The ! function enables shell commands to be integrated into menu selections quickly. Because it specifies a shell command, an application would never be notified when this item is selected. Therefore, you do not need to declare a separate action for it. However, if conditions of the menu item are specified, such as the number of selections or a selection rule, you should declare an action with this information and list the required shell command in the action item arguments.

f.action
The f.action function takes an ActionID, which is associated with some Action declaration elsewhere in the file. It connects the action to the menu item so that when the item is selected, the application is, if necessary, invoked and notified of the selected action.

f.menu
This function provides for the declaration of a menu cascade within a menu. A previous menu example shows f.menu declaring the MailMenu cascade.

f. "built-in-function-name"
Built-in functions are internal to the NetView for AIX program. They are functions that implement internal NetView for AIX callbacks, providing functionality for menu items, such as Help..On Version, File..New Map..., and so on. These are provided so that the NetView for AIX program can use the same file-based menu registration just as other applications do. Each function begins with the f. followed by a unique identifier describing the function. The NetView for AIX registration file contains several examples of internal functions, such as f.star center.

Note: These internal functions are designed for the NetView for AIX program. If you use them with other applications, use them with caution.

Application Action Definition

The Action section is used to define actions which an application can perform. Actions can then be connected to menu items through the registration file or to executable symbols through the user interface.

An application that specifies the Initial flag does not need to define any actions if it will run under the NetView for AIX program. Otherwise, an action must be defined to enable application invocation by menu items or executable symbols.

The action definition contains information, such as the object types that are valid for the action, the command that invokes the application, and the command that passes to the application if it is already running.

Through the EUI API, an application is notified that an action has been requested by registering a callback for the action. This is accomplished using the OVwAddActionCallback registration mechanism. The parameters to the callback will include the name of the action (the ActionID described below), callback arguments specified for the action, and the map and submap where the action was requested. The parameters will also include the current selection list unless the action is invoked from the ObjectMenu, in which case the selection contains the object clicked on to display the ObjectMenu.

ActionID An identifier, or name, for the action that the application can use to receive notification when the action is requested. It is also used as an argument to f.action to tie the action to a menu item. The scope of the name is limited to the application; other applications can have defined actions by the same name.

To continue an example started earlier, the Mail Manager registration referred to an action called MailLog. The following code fragment is a definition for that action that uses elements of the action definition described in the following sections.

```
Application "Mail Manager"
{

    MenuBar "Monitor"
    {
        "Mail"  f.menu "MailMenu";
    }

    Menu "MailMenu"
    {
        "Mail Queue"    f.action "Mailq";
        "Mail Log"      f.action "MailLog";
    }

        :

    Action "MailLog"
    {
        SelectionRule isNode;
        MinSelected 1;
        MaxSelected 1;
        Security;
        Command 'xterm -title "${OVwMenuItem} ($OVwSelection1)" \
                 -e sh -c "/usr/bin/rexec ${OVwSelection1} \
                 -l root tail -f /usr/spool/mqueue/syslog"';
    }

        :
        :
}
```

This action specifies that exactly one object can be selected for the action to take place (MinSelected 1 and MaxSelected 1). It also specifies that the object on which it acts must be a node that supports IP. Finally, it lists the command that should be used to perform the action.

Selection Rule

A logical expression, using the AND (&&), OR (||) and NOT (!) operators, on capability fields. Capability fields are specially designated fields in the object database used for classifying an object. The field registration section below describes how to define a field as a capability.

Capability fields are limited to Boolean and Enumerated types.

The logical expression is classical in its definition. Refer to Grammar on page 784 for the detailed syntax of the SelectionRule expression.

Note: The selection rule applies to the object used to invoke the ObjectMenu when the action is invoked from the ObjectMenu.

MinSelected A means of specifying the minimum number of objects that must be selected for the action to be enabled.

If MinSelected is not specified, it is assumed to be zero, meaning that no objects must be selected for the item to be active.

If MinSelected is not specified, and the action contains a SelectionRule, it is assumed to be 1, meaning that at least one object matching the selection rule must be selected for the action to be activated. If MinSelected is set to zero and the action includes a selection

rule, the action is valid when nothing is selected, but if there are any selections they must meet the selection rule criteria.

When actions are connected to menu items, the value of MinSelected must be consistent across all actions associated with a specific menu item. Two actions (within the same application or from different applications) with different MinSelected settings cannot be connected to the same menu item. Subsequent conflicting menu item registrations are flagged as errors and are ignored.

MaxSelected

An upper boundary for the number of selections on which an action can be applied. If MaxSelected is not specified, any number of objects can be selected for the action. If MaxSelected is set to zero, the action is only valid when nothing is selected.

Security

Specifies that this action is not available unless NetView for AIX security is being used. If security is turned off, the action is still displayed, but it is greyed out and not selectable.

Note: This field is not dynamic, and cannot be checked or changed through the OVwSetAction or OVwGetAction APIs, as can some of the other fields in the Action block.

Process Flags

The same process flags as those for the Application section. In the Action section, these flags override any global (within the application) settings.

A command process is uniquely identified by its flags and command string. In this example:

```
Application "Silly App" {

        Action "Foo"
        {
                Command -Initial -Shared "/usr/bin/foo -x";
        }

                :

        Action Bar
        {
                Command "/usr/bin/foo -x";
        }
}
```

Each command is considered separately because the process flags differ. The Fleep command is shared across actions and is started with the NetView for AIX program. The Bar action command will be invoked each time the action is requested.

Command

The command section provides the same functions as the Command section in the application, but specifies a specific command for an application that can override the application-level command. If no command is specified at the application level, a command must be specified for each defined action. The command in the action section should specify application startup. If the command specifies application startup, the command will start the application if it is not running so that the application can handle the action request. (The application can exit even if the Shared flag is enabled).

Name Field Provides the same object name capability as the NameField setting in the Application section, but on a per-action basis.

CallbackArgs Is a string that is broken into an argument vector and passed to the application's action callback in the argc and argv parameters. It can be used as a more general means for passing parameters from the registration file to specific, application-action callbacks. The CallbackArgs are not available to applications that do not use the EUI API.

ObjectMenu Registration

The ObjectMenu section enables you to register items on the context menu for objects. It works identically to a Menu block. Each item registered in the ObjectMenu block appears on the context menu that appears when you click button 3 on any object in the map.

When you use actions with the context menu, the selection list is set to the object that you clicked on to bring up the context menu, not to the list of selected objects on the map. For example, if you have selected node A and brought up a context menu for node B, the operations you select from the context menu apply to node B instead of node A. Also, the selection list you use to implement action contains node B only. Actions selected from a context menu apply only to that object. You should not use the context menu for actions that require multiple objects.

Tool Registration

The Tool section enables you to register tool items in the Tool window. The following conditions apply to the registration of tool items:

- Tools default to a solid color when no icon is specified.
- SelectionMechanism is mandatory.
- Action is mandatory.
- DragBitmap defaults to a rectangle unless a bitmap icon is used, in which case the drag bitmap defaults to the icon bitmap.
- Position of an item in the Tool window can be determined by setting a precedence for the tool item. (See "OVwAddObjMenuItem(3)" on page 550.)

Application Field Enrollment

The field enrollment section is needed only by applications that will manage the semantics of graphical interface maps. It enables the application to present fields from the object database within dialog boxes used by semantic applications.

Field enrollment is rule based. A rule indicates interest in a particular type of object based on capability fields. The application can enroll various fields based on a rule describing the kind of object associated with a particular dialog box.

The field enrollment section begins with the keyword Enroll, followed by the type of dialog box for which fields are being enrolled. After defining a dialog box with an Enroll block, if you have at least one "Command" statement defined in your registration file, you must register your application to be notified when a user performs actions that require dialog-box input. Your application must register both Query and Confirm callback routines. If you have no "Command" statements defined in your registration file, your Enroll blocks will be accepted by NetView for AIX and will be displayed whenever these actions are invoked by the user.

Dialog Boxes

Applications can enroll fields for the following semantic dialog boxes.

Add	The dialog box presented when adding an object to the map. See "OVwVerifyAdd(3)" on page 833.
Describe	The dialog box presented when describing an object on the map. See "OVwVerifyDescribeChange(3)" on page 849.
Connect	The dialog box presented when connecting two objects on the map. See "OVwVerifyConnect(3)" on page 842.
Configuration	The dialog box presenting per map parameters for configuring the application. This dialog box is special in that it is not associated with a particular object. See "OVwVerifyAppConfigChange(3)" on page 839.

Rule

Rule sections contain field enrollment based on certain capabilities of an object associated with the dialog box. The rule is a logical expression involving capability fields, such as the SelectionRule for application actions. It specifies features of objects the application is interested in for the dialog box.

Note: The rule applies to the object used to invoke the ObjectMenu when the action is invoked from the ObjectMenu.

A dialog box enrollment section can contain several rule sections, each defining a different dialog box. When a dialog box is presented for a particular object, the object is tested against the specified rules. If a rule matches, the field enrollment within the corresponding rule section is used for the dialog box. The first rule that matches (the rules being scanned in the order specified in the registration file) determines the dialog box; no other matching rules are used for the dialog box.

Note: Configuration dialog boxes do not use rules because they are not associated with an object. they contain only field enrollment sections.

Rule Options

The only rule option defined is the InitialVerify option. By default, this option is disabled, or Off. InitialVerify indicates that the application should be immediately contacted when the dialog box is displayed to provide default field values.

Note: If an object exists, as it would for the Describe operation, the field values are initialized automatically to the values set for the object.

Scale

The scale setting provides a scale for parameters to Geometry in the field enrollment section. If no scale is provided, the scale is assumed to be 1.

Field Enrollment

Field enrollment sections specify which fields should appear in the dialog box and how those fields should be presented.

See Example of Field Registration on page 793 for an example field registration file.

Options

The following options can be set per field enrolled. If an option setting is absent, it is assumed to be off.

NoDisplay	If Nodisplay is turned on, it enables the application to be sent the field value even if it is not displayed in the dialog box.
ImmediateVerify	If ImmediateVerify is turned on, send all enrolled fields and their values to the application immediately after a value has been entered in this field.

Label

The label for the field within the dialog box. If no label is specified, the name of the field is used for the label.

Geometry

The Geometry of the field is specified with four integers. They are, in order specified: X-position, Y-position, width, and height.

Edit Policy

The edit policy describes whether the field can be edited or is read only. If the policy is Edit, the field can be edited at any time; if it is NoEdit, the field is only displayed.

Note: Only the first application to enroll a field with the Edit edit policy will be permitted to enroll the field in other dialog boxes with a policy of Edit. Any other application that enrolls this field for a dialog box will automatically get an edit policy of NoEdit.

"EditOnCreation" indicates that the field can be edited only at the time a new map is created. EditOnCreation can be used only for Configuration dialog box fields.

List Display Policy

For fields that are lists, this policy describes how the list is displayed. If a List Display Policy is specified for a field which is not a list, it is ignored.

The only valid ListDisplayPolicy is SelectionListBox, meaning that the list is presented within a selection list. SelectionListBox is the default display policy for lists.

List Selection Policy

The List Selection Policy specifies the behavior of the selection list. Valid choices are:

None
: No specific items can be selected. This is the default.

Single
: Only one item in the selection list can be selected.

Multiple
: More than one item in the selection list can be selected.

Integer Display Policy

The integer display policy describes how Integer32 fields are displayed in the dialog box. Possible choices are:

Integer	The integer is displayed in the usual decimal form. This is the default.
Unsigned	The integer is displayed as an unsigned decimal value.
Hex	The integer is displayed in hexadecimal.
Octal	The integer is displayed in octal.
IPAddr	The integer is displayed as an IP address in dot notation, that is, as the string returned from a call to inet_ntoa(3).

Edit Position

Specifies the position within a String or List type field where editing can occur. By default, if no edit position is specified, the entire field can be edited. The edit position is a range beginning with character x.

Field Default Value

This field enables you to specify a default value for the field. This default value is supported only for application configuration fields.

Symbol Class Registration

A Symbol Class registration file provides a way to define a new class of symbols. See Example of Symbol Class and Symbol Type Registration on page 792 for an example symbol class and symbol type registration file. A symbol class is represented graphically by a symbol's outer shape. The symbol class registration files are located in the directory /usr/OV/symbols/$LANG. These files contain the following information:

Class Name

The name given to the symbol class.

Scale

The scale specifies the scale of coordinates given in the shape description. If no scale is specified, it is assumed to be 1.

Shape

A symbol class shape is defined with Arcs or Segments. The shapes defined by Arc and Segment lines are assumed to be closed polygons.

Arc

Specifies an enclosed Arc shape. It provides a way to display circles, ellipses, and wedges for the class shape. An arc is defined within a conceptual rectangle of a particular width and height. A point is designated within the rectangle as the origin of the end point of a line which will be rotated to form the solid arc. A rotation is given with an optional starting angle and a number of degrees to rotate, meaning that the line whose end point is at the origin, will rotate from the starting angle for the specified number of degrees within the rectangle of the specified width and height.

Origin

An end point for the line that will be rotated to form the arc.

Size

The width and height of the conceptual rectangle in which the line will rotate.

Rotation

Specification of the starting angle and number of degrees to rotate for rotating the line. If no starting angle is specified, it is assumed to be 0, meaning that rotation will start from a three o'clock position within the rectangle. The angles are specified in degrees. The line is rotated from the first angle, which is the number of degrees from a three o'clock position, a total number of degrees specified by the second angle.

Segment

Specifies a series of line segments, which outline a polygon. The last point in the series is automatically connected to the first point in the series.

The *NetView for AIX Programmer's Guide* provides more details on how to construct class shapes with the Segment and Arc statements.

Default Layout

Specifies the default symbol layout algorithm to use for submaps of the symbols in this class. The possible choices are Ring, Bus, Star, Tree, PointToPoint, RowColumn, and None. This default can be overridden within a symbol type definition.

Capabilities

Specifies the default capabilities for objects represented by symbols within this symbol class. These capabilities will be assigned when a object is created for this symbol, that is, when the symbol is placed on the map from the symbol palette.

Capabilities are indicated by specifying a capability field name and its default value. Capability fields are limited to boolean and enumerated types.

These capabilities can be extended or overridden within a symbol type definition.

Variety

Informs the graphical interface what kinds of symbols you are defining. By default, symbols are considered to be ICON symbols, that is, they are icons that represent objects. A symbol class can also contain CONNECTION symbols, meaning that the symbol types represent connections on the map.

Symbol Type Registration

A SymbolType registration file provides a way to define a new symbol type within a class of symbols. A symbol type definition consists of a symbol class, which defines its shape, and bitmaps to put within the class shape. The symbol type registration files are located in the /usr/OV/symbols/$LANG directory.

Class Name

Designates the shape of the symbol. All symbols must be defined within existing symbol classes. If an undefined class name is given, the NetView for AIX program emits an error and ignores the symbol class registration.

Symbol Type Name

Is the name given to the symbol type. A symbol type is then referenced by the combination of its class name and symbol type (subclass) name, for example, Computer:Mainframe.

Symbol Bitmaps (FileBase)

The bitmap for a symbol type appears within the class shape. The basename of the bitmap is specified with the FileBase keyword. The base identifies the bitmap within the /usr/OV/bitmaps/$LANG directory. Files in this directory are named filebase.size.p (the bitmap) and filebase.size.m (the mask). Given the FileBase, the NetView for AIX program searches the bitmap directory for files beginning with this name and determines, from the size portion of the filenames, how many bitmap sizes there are.

CursorSize

Informs the graphical interface which bitmap should be used for the cursor when moving the symbol on the map. The default cursor size is determined automatically by the graphical interface.

Default Layout

Specifies the default symbol layout algorithm to use for submaps of the symbol type. The possible choices are Ring, Bus, Star, Tree, PointToPoint, RowColumn, and None.

Capabilities

Enables you to specify default capabilities for objects represented by this symbol type. These capabilities will be assigned when an object is created for this symbol, that is, when the symbol is placed on the map from the symbol palette.

Capabilities are indicated by specifying a capability field name and its default value. Remember, capability fields are limited to boolean and enumerated types.

Field Registration

The field registration section provides a means for creating fields in the object database. Field registration files can be provided by applications to ensure that fields used within a dialog box enrollment section exist in the object database. Fields cannot be enrolled in a dialog box until they exist in the database.

The field registration section begins with the keyword Field, followed by a block that describes the field's name, type, and properties.

Field Name

The name of the field to be referenced in the object database. The field name must be unique.

Field Type

The following types are allowed for fields in the object database.

Boolean — True or False value.

String — Any character string.

Enumeration
: An enumerated type. The specific enumeration constants can be declared in an enumeration section of the field registration.

Integer32 — A 32 bit integer.

Field Flags

The field flags indicate properties of the field. They are:

List — A list of the specified type. The only supported types for lists are strings and integers.

Name — A name field. Name fields uniquely identify objects, so there is only one object with a specific value for this field.

Locate — A locate operation can be performed on this field. This field will appear in the Locate by Attribute dialog box.

General — Appears in the general attributes section of the Add and Description dialog boxes on an object.

Capability — A capability field, used to classify an object. Only booleans and enumerated types are supported as capability fields.

Field Enumeration

This section specifies the constant symbolic names of the enumerated values for an enumerated type. The first name listed is the name for the value, or index, 0 (zero); the second is the name for the value, or index, 1; and so on. The first value must be the name unset.

Grammar

The following example is a consolidated grammar for application, symbol, and field registration files. Elements of the grammar that are in quotation marks are tokens that are recognized case insensitively.

```
Registration ::= Application | Symbols | Fields | Empty

Application ::= "Application" AppName AppParent AppBlock
```

```
AppName ::= StringOrIdentifier

StringOrIdentifier ::= String | Identifier

String ::= “\”” zero or more characters “\”” |
           “’” zero or more characters “’”

Identifier ::= one letter (a-z) followed by zero or more letters,
               digits, or underscores

AppParent ::= “:” AppName | Empty

AppBlock ::= EmptyBlock | “{” AppStmts “}”

EmptyBlock ::= “{” “}”

AppStmts ::= AppStmt | AppStmts AppStmt

AppStmt ::= Description | HelpDirectory | Naming | Command | Version
          | Copyright | MenuBar | ObjectMenu | Tool | Menu | Action
          | Enrollment | “;”

Description ::= “Description” DescriptionBlock

DescriptionBlock ::= EmptyBlock | “{” DescriptionText “}”

DescriptionText ::= Strings

Strings ::= String | String “,” | String “,” Strings

Copyright ::= “Copyright” CopyrightBlock

CopyrightBlock ::= EmptyBlock | “{” CopyrightText “}”

CopyrightText ::= Strings

HelpDirectory ::= “HelpDirectory” Pathname “;”

Pathname ::= String

Naming ::= “NameField” NameFields “;”

NameFields ::= NameField | NameFields “,” NameField

NameField ::= StringOrIdentifier

Command ::= “Command” ProcessFlags String “;” | “Command” String “;”

ProcessFlags ::= ProcFlag | ProcessFlags ProcFlag

ProcFlag ::= “-Initial” | “-Shared” | “-Restart”

Version ::= “Version” String “;”

MenuBar ::= “MenuBar” Precedence MenuID Mnemonic MenuBlock

Mnemonic ::= An underscore followed by one or more non-whitespace characters
```

```
Integer ::=  one or more decimal digits
             | "-" one or more decimal digits

Menu ::= "Menu" MenuID MenuBlock

MenuID ::= StringOrIdentifier

ObjectMenu ::= "ObjectMenu" MenuBlock

Tool ::= "Tool" Precedence ToolID ToolBlock

ToolBlock ::= "{" ToolStmts "}"

ToolID ::= StringOrIdentifier

ToolStmts ::= ToolStmt | ToolStmts ToolStmt

ToolStmt ::=
   Icon
   | LabelColor |DragBitmap |SelectionMechanism.
   | ActionSpecification | ";"

Icon ::= "Icon" IconSpecifier StringOrIdentifier

IconSpecifier ::=  "Bitmap"
   | "Gif" | "Solid"

LabelColor ::=  "LabelColor" StringOrIdentifier

DragBitmap ::=  "DragBitmap" StringOrIdentifier

SelectionMechanism ::= "SelectionMechanism" SelectionSpecifierList

SelectionSpecifierList ::=
   SelectionSpecifierList "," SelectionSpecifier | SelectionSpecifier

SelectionSpecifier ::= "double-click"    | "drag-drop"

MenuBlock ::= EmptyBlock | "{" MenuStmts "}"

MenuStmts ::= MenuItem | MenuStmts MenuItem

MenuItem ::= Precedence Label Mnemonic Accelerator Function ";" | ";"

Precedence ::= "<" Integer ">" | Empty

Label ::= String | Bitmap

Bitmap ::= "@" Pathname

Accelerator ::= ModifierList "<Key>" KeyName
                | Empty

ModifierList ::= ModifierName |  ModifierList ModifierName

ModifierName ::=  "Ctrl" | "Shift" | "Alt" | "Meta" | "Lock"
                  | "Mod1" | "Mod2" | "Mod3" | "Mod4" | "Mod5"
                  | "None" | "Any"
```

```
KeyName ::=  An X11 keysym name. Keysym names can be in the
             keysymdef.h file (remove the XK_ prefix).

Function ::= "!" ShellCommand | "f.action" FunctionArg
             | "f.menu" FunctionArg

FunctionArg ::= StringOrIdentifier | Empty

ShellCommand ::= String

Action ::= "Action" ActionID ActionBlock

ActionID ::= StringOrIdentifier

ActionBlock ::= EmptyBlock | "{" ActionStmts "}"

ActionStmts ::= ActionStmt | ActionStmts ActionStmt

ActionStmt ::= SelectionRule | Naming | MinSelected | MaxSelected
               | CallbackArgs | Command | ";"

SelectionRule ::= "SelectionRule" Expression ";"

Expression ::= Expression "&&" Expression | Expression "||" Expression
               | "!" Expression | "(" Expression ")" | BooleanExpression

BooleanExpression ::= Integer | CapabilityFieldName
                      | CapabilityFieldName Operator String
                      | CapabilityFieldName Operator Integer

CapabilityFieldName ::= StringOrIdentifier

Operator ::= "==" | "!="

MinSelected ::= "MinSelected" Integer ";"

MaxSelected ::= "MaxSelected" Integer ";"

CallbackArgs ::= "CallbackArgs" String

Enrollment ::= "Enroll" AddDescribe | "Enroll" Connect | "Enroll" Config

AddDescribe ::= "Add" AddDescribeBlock | "Describe" AddDescribeBlock
                | "Add" "," "Describe" AddDescribeBlock
                | "Describe" "," "Add" AddDescribeBlock

AddDescribeBlock ::= EmptyBlock | "{" EnrollAddRules "}"

EnrollAddRules ::= Rule | EnrollAddRules Rule

Connect ::= "Connect" ConnectBlock

ConnectBlock ::= EmptyBlock | "{" EnrollConRules "}"

EnrollConRules ::= ConnectRule | EnrollConRules ConnectRule

Config ::= "Configuration" ConfigBlock
```

```
ConfigBlock ::= EmptyBlock | "{" FieldEnrolls "}"

Rule ::= "If" Expression FieldEnrollsBlock

FieldEnrollsBlock ::= EmptyBlock | "{" FieldEnrolls "}"

FieldEnrolls ::= FieldEnrollment | FieldEnrolls FieldEnrollment

RuleGlobal ::= Scale | RuleOptions | EnrollHelp | ";"

EnrollHelp ::= "HelpFile" Pathname ";" | "HelpIndex" Pathname ";"

ConnectRule ::= "If" Expression "," Expression FieldEnrollsBlock

RuleOptions ::= "InitialVerify" OnOff ";"

OnOff ::= "On" | "Off"

FieldEnrollment ::= RuleGlobal | "Field" FieldEnrollName FieldEnrollmentBlock

FieldEnrollName ::= StringOrIdentifier

FieldEnrollmentBlock ::= EmptyBlock | "{" FieldEnrollStmts "}"

FieldEnrollStmts ::= FieldEnrollStmt | FieldEnrollStmts FieldEnrollStmt

FieldEnrollStmt ::= FieldOptions | FieldLabel | FieldGeometry
                    | FieldEditPolicy | FieldListDisplayPolicy
                    | FieldListSelectionPolicy | FieldIntegerDisplayPolicy
                    | FieldEditPosition | FieldDefaultValue | ";"

FieldOptions ::= "NoDisplay" OnOff ";" | "ImmediateVerify" OnOff ";"

FieldLabel ::= "Label" String ";"

FieldGeometry ::= "Geometry" Integer "," Integer ";"
          | "Geometry" Integer "," Integer "," Integer "," Integer ";"

FieldEditPolicy ::= "EditPolicy" EditPolicy ";"

EditPolicy ::= "Edit" | "NoEdit" | "EditOnCreation"

FieldListDisplayPolicy ::= "ListDisplayPolicy" ListDisplayPolicy ";"

ListDisplayPolicy ::= "SelectionListBox"

FieldListSelectionPolicy ::= "ListSelectionPolicy" ListSelectionPolicy ";"

ListSelectionPolicy ::= "None" | "Single" | "Multiple"

FieldIntegerDisplayPolicy ::= "IntegerDisplayPolicy" IntegerDisplayPolicy ";"

IntegerDisplayPolicy ::="Hex" | "Octal" | "Integer" | "Unsigned" | "IPAddr"

FieldEditPosition ::= "EditPosition" Integer ";"

FieldDefaultValue ::= "DefaultValue" StringOrIdentifier ";"
```

```
Symbols ::= Symbol | Symbols Symbol

Symbol ::= SymbolClass | SymbolType

SymbolClass ::= “SymbolClass” SymClassName SymClassBlock

SymClassName ::= StringOrIdentifier

SymClassBlock ::= EmptyBlock | “{” SymClassStmts “}”

SymClassStmts ::= SymClassStmt | SymClassStmts SymClassStmt

SymClassStmt ::= Scale | Shape | VarietyStmt | Capabilities
                 | DefaultStatusSource | Layout | “;”

Scale ::= “Scale” Integer “;”

Shape ::= Arc | Segment

Arc ::= “Arc” Origin Size Rotation “;”

Origin ::= “Origin” Point

Point ::= “(” Integer “,” Integer “)”

Size ::= “Size” Point

Rotation ::= “Rotation” Integer | “Rotation” Integer “,” Integer

Segment ::= “Segment” Segments “;”

Segments ::= Point | Point “To” Segments | Empty

VarietyStmt ::= “Variety” Variety “;”

Variety ::= “Icon” | “Connection”

DefaultStatusSource ::= “DefaultStatusSource” StatusSource “;”

StatusSource ::= “Compound” | “Object” | “Symbol”

SymbolType ::= “SymbolType” SymClassName “:” SymbolName SymBlock

SymbolName ::= StringOrIdentifier

SymBlock ::= EmptyBlock | “{” SymStmts “}”

SymStmts ::= SymStmt | SymStmts SymStmt

SymStmt ::= FileBase | CursorSize | Layout | Capabilities
            | DefaultStatusSource | LineStmts | “;”

FileBase ::= “FileBase” StringOrIdentifier “;”

CursorSize ::= “CursorSize” Integer

LineStmts ::= LineStyle | LineDashPattern
```

```
LineStyle ::= “LineStyle” “Solid” “;” | “LineStyle” “Dash” “;”

LineDashPattern ::= “LineDashPattern” DashLengths “;”

DashLengths ::= DashLength | DashLength “,” DashLengths

DashLength ::= Integer

Layout ::= “DefaultLayout” LayoutName “;”

LayoutName ::= “Ring” | “Bus” | “Star” | “Tree” |
“PointToPoint” | “RowColumn”

Capabilities ::= “Capabilities” CapabilityBlock

CapabilityBlock ::= EmptyBlock | “{” CapabilityStmts “}”

CapabilityStmts ::= CapabilityStmt | CapabilityStmts CapabilityStmt

CapabilityStmt ::= CapabilityFieldName “;”
                   | CapabilityFieldName “=” StringOrIdentifier “;”
                   | CapabilityFieldName “=” Integer “;”

Fields ::= Field | Fields Field

Field ::= “Field” FieldRegisterName FieldBlock

FieldRegisterName ::= StringOrIdentifier

FieldBlock ::= EmptyBlock | “{” FieldDefStmts “}”

FieldDefStmts ::= FieldDefStmt | FieldDefStmts FieldDefStmt

FieldDefStmt ::= FieldType | FieldFlags | FieldEnumeration | “;”

FieldType ::= “Type” FieldTypeName “;”

FieldTypeName ::= “Boolean” | “String” | “Enumeration” | “Integer32”

FieldFlags ::= “Flags” FieldFlagStmts “;”

FieldFlagStmts ::= FieldFlag | FieldFlag “,” FieldFlagStmts | Empty

FieldFlag ::= “List” | “Name” | “Locate” | “General” | “Capability”

FieldEnumeration ::= “Enumeration” EnumDefs “;”

EnumDefs ::= EnumDef | EnumDef “,” EnumDefs

EnumDef ::= EnumName

EnumName ::= StringOrIdentifier

Empty ::=
```

Examples

The following examples are simple registration files for the registrations described in the previous sections.

Example of Application Registration

```
/*  XYZ Windows Terminal Connect
*/

Application “XYZ Windows Terminal Connect”
{
    Version “2.0”;

    Description {
        “Enables you to create a terminal emulator window on a local ”,
        “system that is connected to a remote system by a network ”,
        “virtual terminal protocol. ”
    }

    Copyright {
      “(c)Copyright 1992 XYZ Corp..”
     }


    // There is no application command statement.  Instead, each action
    // defines its own command.


    MenuBar <100> “Misc” _s
    {
        <100> “Terminal Connect”   _T   f.menu “Terminal Connect”;
    }

    Menu “Terminal Connect”
    {
        <100> “Telnet (xterm)”  _x f.action “XTerm Telnet”;
        <100> “Vt3k”   _V f.menu   “Vt3k Menu”;
    }

    Menu “Vt3k Menu”
    {
        <100> “Block Mode ...” _B f.action “Block vt3k”;
        <100> “Type Ahead ...” _T f.action “Typeahead vt3k”;
    }

    Action “XTerm Telnet”
    {
        MinSelected 1;
        MaxSelected 1;
        SelectionRule isNode&& (vendor==“IBM”);
        NameField “IP Hostname”;
        Command “${xnmtelnet:-/usr/OV/bin/xnmtelnet} xterm”;
    }

    Action “Block vt3k”
    {
```

```
        MinSelected 1;
        MaxSelected 1;
        SelectionRule isNode && (vendor==“IBM”);
        NameField “IP Hostname”;
        Command “${xnmvt3k:-/usr/OV/bin/xnmvt3k} block”;
    }
    Action “Typeahead vt3k”
    {
        MinSelected 1;
        MaxSelected 1;
        SelectionRule isNode && (vendor==“XYZ”);
        NameField “IP Hostname”;
        Command “${xnmvt3k:-/usr/OV/bin/xnmvt3k}typeahead”;
    }
}
```

Example of Symbol Class and Symbol Type Registration

```
/* Network symbols */

SymbolClass “Network”
{
        Scale 7;
        Arc Origin (-2, -1) Size (17, 17) Rotation 0, 360;
        DefaultStatusSource Compound;
        DefaultLayout PointToPoint;
        Variety Icon;
        Capabilities {
                isNetwork = 1;
        }
}

SymbolType “Network” : “Network”
{
        Filebase “ip_net”;
        CursorSize 38;
}

SymbolType “Network” : “IP Network”
{
        Filebase “ip”;
        CursorSize 38;

        Capabilities {
                isIP = 1;
        }
}

SymbolType “Network” : “Bus”
{
        Filebase “bus”;
        CursorSize 38;

        Capabilities {
                isNetwork = 0;
                isSegment = 1;
```

```
                isBusSegment = 1;
        }

        DefaultLayout Bus;
}

SymbolType “Network” : “Token Ring”
{
        Filebase “ring”;
        CursorSize 38;

        Capabilities {
                isNetwork = 0;
                isSegment = 1;
                isTokenRingSegment = 1;
        }

        DefaultLayout Ring;
}
```

Example of Field Registration

```
/* OVW fields */

Field “vendor” {
        Type    Enumeration;
        Flags   capability, general, locate;
        Enumeration “Unset”, “IBM”, “Other”;
}

Field “IP Hostname” {
        Type    StringType;
        Flags   name, locate;
}

Field “isLocation” {
        Type    Boolean;
        Flags   capability;
}

Field “isIP” {
        Type    Boolean;
        Flags   capability, locate;
}
```

Libraries

When compiling a program that uses NetView for AIX graphical interface registration files, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See ovwdb(8).
- See *NetView for AIX Application Interface Style Guide*.
- See *NetView for AIX Programmer's Guide*.
- See *NetView for AIX User's Guide for Beginners*.

OVwRenameRegContext(3)

Purpose

Changes the name of a registration context

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwRenameRegContext(char *fromAppName, char *toAppName);
```

Description

OVwRenameRegContext changes the name of a registered NetView for AIX application.

Before calling OVwRenameRegContext, the application must have successfully called OVwLockRegUpdates to acquire permission to modify the registration context. Changes to the application registration will only become permanent after calling OVwSaveRegUpdates.

Use this function if your application needs to dynamically create or modify application registration. If your application registration is static, use the application registration files for defining registration information.

Parameters

fromAppName	Specifies a pointer to the name of the application that is being altered. If fromAppName is NULL, the current application is renamed.
toAppName	Specifies a pointer to the new application name.

Return Values

If successful, OVwRenameRegContext returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwRenameRegContext sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	The application appName is not a registered application.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_PERMISSION_DENIED]	The application has not called OVwLockRegUpdates prior to this function call.

Implementation Specifics

OVwRenameRegContext supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwRenameRegContext, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetAppConfigValues(3)” on page 698.
- See “OVwGetFirstRegContext(3)” on page 717.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwSaveRegUpdates(3)

Purpose

Saves modifications to registration information

Related Functions

OVwUndoRegUpdates

Syntax

```
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwSaveRegUpdates(OVwBoolean updateFiles);

int OVwUndoRegUpdates();
```

Description

OVwSaveRegUpdates causes the NetView for AIX program to save changes that have been made to registration information through the EUI API. The changes affect the current NetView for AIX session and can optionally be saved in the appropriate application registration files. EUI API calls that alter application registration information include OVwActionRegistration, OVwAddMenuItem, OVwAddMenuItemFunction, OVwAppRegistration, OVwMenuRegistration, and OVwMenuItemRegistration.

OVwUndoRegUpdates will destroy any changes that have been made to registration information since the last call to OVwSaveRegUpdates or OVwLockRegUpdates.

Parameters

updateFiles If TRUE, changes to registration information are saved in the appropriate application registration files. If FALSE, only the current graphical interface session will retain the modifications.

Return Values

If successful, OVwSaveRegUpdates and OVwUndoRegUpdates return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwSaveRegUpdates and OVwUndoRegUpdates set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED] The application either has not called OVwLockRegUpdates prior to this function call or is not running with a uid or gid with permission to save changes to the registration files.

Libraries

When compiling a program that uses OVwSaveRegUpdates or OVwUndoRegUpdates, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwLockRegUpdates(3)” on page 755.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwSelectListChangeCB(3)

Purpose

Functions as a callback for a selection list change event

Syntax

```
#include <OV/ovw.h>

void (*OVwSelectListChangeCB) (void *userData, OVwEventType type,
     OVwMapInfo *map);
```

Description

To receive an event indicating that the user has changed the map selection list, use OVwAddCallback to register a callback function of type OVwSelectListChangeCB to be called when an ovwSelectListChange event is generated.

Parameters

map Specifies a pointer to the OVwMapInfo structure for the map whose selection list has changed.

type Specifies the type of event that caused this callback to be invoked. For this callback, type will always be ovwSelectListChange.

userData Specifies a pointer to the application data registered by the OVwAddCallback function.

Examples

The following code fragment shows an example of registering a callback routine for receiving a map selection list change event:

```
void
SelectListChangeProc(void *userData, OVwEventType type,
        OVwMapInfo *map)
{
    OVwObjectIdList *objs;

    /* Get the new selection list */
    objs = OVwGetSelections(map, NULL);

    /* Highlight the new selections */
    OVwHighlightObjects (map, objs, FALSE);
}

OVwAddCallback(ovwSelectListChange, NULL,
                (OVwCallbackProc) SelectListChangeProc, NULL);
```

Implementation Specifics

OVwSelectListChangeCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSelectListChangeCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwSetBackgroundGraphic(3)

Purpose

Sets the background graphic for a submap

Related Functions

OVwClearBackgroundGraphic

Syntax

```
#include <OV/ovw.h>

int OVwSetBackgroundGraphic(OVwMapInfo *map, OVwSubmapId submapId,
          char *filename);

int OVwClearBackgroundGraphic(OVwMapInfo *map, OVwSubmapId submapId);
```

Description

OVwSetBackgroundGraphic is used to set the background picture for a specified submap.

OVwClearBackgroundGraphic is used to remove a picture from the specified submap and restore the standard mapBackground color as a solid pattern.

Parameters

filename Specifies a pointer to the fully-qualified name of the file that contains the picture information. The picture can be in one of the following 2 formats:

- Graphics Interchange Format (GIF) by CompuServe**
- X Bitmap format.

The distinction between picture formats is made automatically.

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

submapId Specifies the ID of the submap.

Return Values

If successful, OVwSetBackgroundGraphic and OVwClearBackgroundGraphic return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwSetBackgroundGraphic and OVwClearBackgroundGraphic set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_BG_FILE_ERROR] The file specified by filename was not found.

[OVw_BG_BAD_FORMAT] The file specified by filename is not in a valid graphics format.

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open read-only.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap specified by submapId does not exist on the open map.
[OVw_SUBMAP_PERMISSION_DENIED]	The submap is an exclusive submap created by another application.

Implementation Specifics

OvwSetBackgroundGraphic and OVwClearBackgroundGraphic support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetBackgroundGraphic or OVwClearBackgroundGraphic, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- CompuServe Graphics Interchange Format(sm) Version 89a.

OVwSetStatusOnObject(3)

Purpose

Sets status of objects and symbols

Related Functions

OVwSetStatusOnObjects
OVwSetStatusOnSymbol
OVwSetStatusOnSymbols

Syntax

```
#include <OV/ovw.h>

int OVwSetStatusOnObject(OVwMapInfo *map, OVwObjectId objectId,
          OVwStatusType status);

int OVwSetStatusOnObjects(OVwMapInfo *map,
          OVwObjectStatusList *objectList);

int OVwSetStatusOnSymbol(OVwMapInfo *map, OVwSymbolId symbolId,
          OVwStatusType status);

int OVwSetStatusOnSymbols(OVwMapInfo *map,
          OVwSymbolStatusList *symbolList);
```

Description

OVwSetStatusOnObject sets the status of all symbols on the open map of the object specified by objectId that have the symbol status source ovwObjectStatusSource. No error is returned if none of the symbols representing the object has status source ovwObjectStatusSource. In addition to being set on all symbols whose status source is ovwObjectStatusSource, the object status is stored in the object_status field of the OVwObjectInfo structure for the object.

OVwSetStatusOnObjects sets the object status on multiple objects. This is more efficient than making separate calls to set the status for each object, because compound status propagation is disabled until the status of all objects in the list have been set. OVwSetStatusOnObjects will return an error if the operation fails for any of the items in the list. Even if an error occurs, the operation will still be performed for all those items on which it can. Upon return, the error field of the OVwObjectStatus structure indicates whether the operation succeeded for a particular object; a value of OVw_SUCCESS indicates success.

OVwSetStatusOnSymbol sets the status of the symbol specified by symbolId to the given status if the symbol has status source, ovwSymbolStatusSource, and if the application has permission to modify the symbol. The application has permission to modify the symbol if it is not a symbol on the application plane of an exclusive submap created by another application. An error results if the status source of the symbol is not ovwSymbolStatusSource or the application does not have permission.

OVwSetStatusOnSymbols sets the symbol status on multiple symbols. This is more efficient than making individual calls to set the status for each symbol, because compound status propagation is disabled until the status of all symbols in the list have been set. OVwSetStatusOnSymbols will return an error if the operation fails for any of the items in the list. Even if an error occurs, the operation will still be performed

for all those items on which it can. Upon return, the error field of the OVwObjectStatus structure indicates whether the operation succeeded for a particular symbol; an OVw_SUCCESS value indicates success.

The OVwSetSymbolStatusSource routine can be used to change the status source of a symbol. Valid status source values are ovwSymbolStatusSource, ovwObjectStatusSource, and ovwCompoundStatusSource.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
objectId	Specifies the ID of the object.
objectList	Specifies a pointer to a list of objects and the status values to set.
status	Specifies the status. The permitted values are defined in the <OV/ovw_types.h> header file:
	ovwUnknownStatus — The status is unknown.
	ovwNormalStatus — The status is up or normal.
	ovwMarginalStatus — The status is marginal (some problem exists).
	ovwCriticalStatus — The status is down or critical.
	The value ovwUnmanagedStatus is not valid and cannot be used to unmanage an object; unmanaging objects is solely under end user control.
	The value ovwAcknowledgeStatus can be set by OVwSetStatusOnObject or OVwSetStatusOnObjects but cannot be set by OVwSetStatusOnSymbol or OVwSetStatusOnSymbols.
symbolId	Specifies the ID of the symbol.
symbolList	Specifies a pointer to a list of symbols and the status values to set for them.

Return Values

If successful, OVwSetStatusOnObject and its related functions return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwSetStatusOnObject and its related functions set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_OBJECT_INVALID_STATUS]	The argument status has a value that is not valid.
[OVw_OBJECT_NOT_ON_MAP]	The object specified by objectId does not exist on the open map.

[OVw_OBJECT_UNMANAGED]	The object indicated by objectId or symbolId is unmanaged. The status of an unmanaged object cannot be set.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SYMBOL_INVALID_STATUS]	The argument status has a value that is not valid.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.
[OVw_SYMBOL_STATUS_SOURCE_MISMATCH]	The symbol, specified by symbolId, does not have status source, ovwSymbolStatusSource.
[OVw_SUBMAP_PERMISSION_DENIED]	The symbol, specified by symbolId, has status source, ovwSymbolStatusSource, and the symbol is on the application plane of an exclusive submap created by another application.

Implementation Specifics

OVwSetStatusOnObject and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetStatusOnObject or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwSetSymbolStatusSource(3)” on page 820.
- See “OVwApiIntro(5)” on page 560.

OVwSetSubmapName(3)

Purpose

Sets name of a submap

Syntax

```
#include <OV/ovw.h>

int OVwSetSubmapName(OVwMapInfo *map, OVwSubmapId submapId,
          char *submapName);
```

Description

OVwSetSubmapName sets the name of a submap. The name of a submap usually identifies the parent object of the submap. Although the submap name is not required to be unique, it is good practice for it to be a unique name. The submap name is displayed as the title of the submap window and is used to identify the submaps in the map dialog box.

An application is not permitted to change the name of an exclusive submap created by another application.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
submapId	Specifies the submap ID of the submap.
submapName	Specifies a pointer to the name of the submap.

Return Values

If successful, OVwSetSubmapName returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwSetSubmapName sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open with read-only permission.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_NOT_FOUND]	The submap specified by submapId does not exist on the open map.
[OVw_SUBMAP_PERMISSION_DENIED]	The submap is an exclusive submap created by another application.

Implementation Specifics

OVwSetSubmapName supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSubmapName, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwSetSymbolApp(3)

Purpose

Sets application interest in a symbol

Related Functions

OVwClearSymbolApp

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolApp(OVwMapInfo *map, OVwSymbolId symbolId);

int OVwClearSymbolApp(OVwMapInfo *map, OVwSymbolId symbolId);
```

Description

OVwSetSymbolApp is used to express application interest in a symbol. It adds the calling application to the list of applications interested in the symbol. This list is stored in the apps field of the OVwSymbolInfo structure for the symbol. The application that creates a symbol is automatically added to the list of interested applications for that symbol. Also, any application the user configures for that symbol is automatically added to the list of interested applications for that symbol.

This method enables applications to define a set of symbols on a particular submap that satisfy the semantic constraints for that submap. OVwListSymbols can be used to get a list of only those symbols on a particular submap in which an application is interested.

OVwClearSymbolApp is used to clear application interest in a symbol. It removes the calling application from the list of applications interested in the symbol. An application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
symbolId	Specifies the ID of the symbol.

Return Values

If successful, OVwSetSymbolApp and OVwClearSymbolApp return 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwSetSymbolApp and OVwClearSymbolApp set the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open with read-only permission.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_PERMISSION_DENIED]	The symbol exists on the application plane of an exclusive submap created by another application.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwSetSymbolApp and OVwClearSymbolApp support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolApp or OVwClearSymbolApp, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwListSymbols(3)” on page 750.
- See “OVwApiIntro(5)” on page 560.

OVwSetSymbolBehavior(3)

Purpose

Sets the behavior of a symbol

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolBehavior(OVwMapInfo *map, OVwSymbolId symbolId,
      int behavior, char *appName, char *actionId,
      OVwObjectIdList *targetObjects);
```

Description

OVwSetSymbolBehavior sets the behavior of a symbol on the open map to either ovwSymbolExplodable or ovwSymbolExecutable. By default, symbols are created as explodable. OVwSetSymbolBehavior must be used to make a symbol executable.

Double-clicking with the mouse on an explodable symbol results in the display of a child submap that shows the contents of the object that is represented by the symbol, if such a submap exists.

Double-clicking with the mouse on a executable symbol results in an assigned action being performed by an application, using a pre-set list of target objects as the selection list for the action.

If ovwSymbolExecutable is specified as the symbol behavior, the appName parameter specifies the application whose application registration file contains a definition of the action, specified by actionId, to perform when the symbol is executed. An error occurs if the application or action are not registered. See “OVwRegIntro(5)” on page 769 for more information on application registration. The application can register to receive a callback for the registered action using OVwAddActionCallback.

Changing the behavior of a symbol to executable does not affect how the symbol gets its status. If an executable symbol has compound status source, it still derives its status from the child submap of its associated object. The existence of a child submap is not affected by making a symbol executable. The child submap will continue to exist, even though it can no longer be accessed through this particular symbol. If the last explodable symbol of a parent object is made executable, the only way to access the child submap is through the submap list box.

An application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

Parameters

actionId Specifies a pointer to the action to perform when the symbol is executed. The action specified by actionId must be defined in an application registration file for the application appName. This parameter is used only if behavior is ovwSymbolExecutable.

appName Specifies a pointer to the application that has registered the action specified by actionId. OVwGetAppName returns the name of the calling application. This parameter is used only if behavior is ovwSymbolExecutable.

behavior Specifies the symbol behavior. The permitted values are defined in <OV/ovw.h>:

ovwSymbolExplodable
The symbol opens into the child submap of its object.

ovwSymbolExecutable
The symbol can execute an application action.

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbolId Specifies the ID of the symbol.

targetObjects Specifies a pointer to a list of objects on which to perform the action actionId. This object list is used as the selection list sent to the application when the action is performed by double-clicking on the executable symbol. If this argument is NULL, the selection list will contain no objects. This parameter is used only if behavior is ovwSymbolExecutable.

Return Values

If successful, OVwSetSymbolBehavior returns 0 (zero). If unsuccessful, they return –1 (negative one).

Error Codes

OVwSetSymbolBehavior sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_ACTION_NOT_APPLICABLE] This error code is returned if any of the following conditions are true.

- The objects specified by targetObjects do not meet the selection list requirements of the action specified by actionId. See the registration file definition of the action to find its selection list requirements.
- There is no command specifying the action.
- The command does not match the application's command.

[OVw_ACTION_NOT_FOUND] The action specified by actionId is not registered by the application appName.

[OVw_APP_NOT_FOUND] The application appName is not a registered application.

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_MAP_READ_ONLY] The map is open with read-only permission.

[OVw_OBJECT_NOT_ON_MAP] An object specified in targetObjects does not exist on the open map.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_SUBMAP_PERMISSION_DENIED] The symbol exists on the application plane of an exclusive submap created by another application.

[OVw_SYMBOL_INVALID_BEHAVIOR]	The argument behavior has a value that is not valid.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwSetSymbolBehavior supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolBehavior, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddActionCallback(3)” on page 532.
- See “OVwError(3)” on page 688.
- See “OVwGetAppName(3)” on page 701.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwSetSymbolLabel(3)

Purpose

Sets label of a symbol

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolLabel(OVwMapInfo *map, OVwSymbolId symbolId,
       char *label);
```

Description

OVwSetSymbolLabel sets the label of a symbol on the open map. The symbol label is displayed beneath the symbol and does not need to be unique.

OVwSetSymbolLabel should be used with discretion. If an application originally set the symbol label and the user has not modified it (that is, the label has the original value set by the application), the application can change the label. Generally, an application should not change a label that has been modified by the user.

An application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

Parameters

label
Specifies a pointer to the new symbol label.

map
Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbolId
Specifies the ID of the symbol.

Return Values

If successful, OVwSetSymbolLabel returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwSetSymbolLabel sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open with read-only permission.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_PERMISSION_DENIED]	The symbol exists on the application plane of an exclusive submap created by another application.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwSetSymbolLabel supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolLabel, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwSetSymbolPosition(3)

Purpose

Sets position of a symbol

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolPosition(OVwMapInfo *map, OVwSymbolId symbolId,
     OVwSymbolPosition *position);
```

Description

OVwSetSymbolPosition moves a symbol to a new position. The symbol identified by symbolId is moved to the position specified by the argument position. Only an icon symbol (a symbol with the variety ovwIconSymbol) can be moved using OVwSetSymbolPosition. The effect of moving a symbol depends on the placement style used, the layout style of the submap on which the symbol exists, whether automatic layout is enabled or disabled on the submap, and whether the symbol is hidden.

The effect of using the position parameter of the symbol creation routines is the same as using OVwSetSymbolPosition, except that a newly created symbol has no previous position from which to be moved. See “OVwCreateSymbol(3)” on page 623 for more information.

An application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

The placement field of the position argument specifies the placement style to use when moving the symbol. Certain placement values are valid for certain types of submap layouts. The placement styles ovwNoPosition and ovwCoordPosition are valid for all layouts. (A NULL value for the position argument is equivalent to specifying ovwNoPosition as the placement style.) The placement style ovwSequencePosition is valid for any sequence layout (that is, ovwRowColumnLayout, ovwBusLayout, ovwStarLayout, ovwRingLayout, and ovwTreeLayout). The placement style ovwStarCenterPosition is valid only for the star layout, ovwStarLayout.

The following list summarizes the valid placement styles for each submap layout:

Submap layout	Placement styles
ovwNoLayout	ovwNoPosition, ovwCoordPosition
ovwPointToPointLayout	ovwNoPosition, ovwCoordPosition
ovwRowColumnLayout	ovwNoPosition, ovwCoordPosition, ovwSequencePosition
ovwBusLayout	ovwNoPosition, ovwCoordPosition, ovwSequencePosition
ovwStarLayout	ovwNoPosition, ovwCoordPosition, ovwSequencePosition, ovwStarCenterPosition
ovwRingLayout	ovwNoPosition, ovwCoordPosition, ovwSequencePosition
ovwTreeLayout	ovwNoPosition, ovwCoordPosition, ovwSequencePosition
ovwMultipleConnLayout	ovwNoPosition, ovwCoordPosition

The position field of the OVwSymbolInfo structure gives the current position of a given icon symbol. The following list summarizes the placement style information returned for symbols in submaps having each layout:

Submap layout	Placement styles
ovwNoLayout	ovwNoPosition, ovwCoordPosition
ovwPointToPointLayout	ovwNoPosition, ovwCoordPosition
ovwRowColumnLayout	ovwNoPosition, ovwSequencePosition
ovwBusLayout	ovwNoPosition, ovwSequencePosition
ovwStarLayout	ovwNoPosition, ovwSequencePosition, ovwStarCenterPosition
ovwRingLayout	ovwNoPosition, ovwSequencePosition
ovwTreeLayout	ovwNoPosition, ovwSequencePosition
ovwMultipleConnLayout	ovwNoPosition, ovwCoordPosition

No Position: If the placement is specified as ovwNoPosition or a value of NULL is used for the position argument, the symbol is first removed from its previous position. If automatic layout is enabled for the submap, the layout algorithm is then executed to place the symbol. If automatic layout is disabled, the symbol is added to the symbol holding area at the bottom of the submap for future inclusion in the submap. Symbols in the holding area always have a placement value of ovwNoPosition.

Coordinate Position: If the placement is specified as ovwCoordPosition, the symbol is removed from its previous position and placed at the specified coordinate position. Symbol placement by coordinates takes effect immediately, regardless of whether automatic layout is enabled or disabled.

The width and height fields of the OVwSymbolPosition structure are used to define a grid coordinate system for interpreting the X and Y symbol coordinates. For example, a symbol placed at position (100, 100) on a (200 x 200) grid will be placed in the center of the submap. A symbol placed at position (150, 200) on a (300 x 400) grid will also be placed in the center of the submap. For best results, use the same grid size for all symbols placed on a particular submap. The grid coordinates are automatically translated into the virtual coordinates that are stored for the submap; the virtual coordinates are, in turn, translated into screen coordinates used when the symbol is actually displayed. It is not necessary to know the virtual size of a submap or its screen size when placing symbols, because symbols are placed according to a grid coordinate system specified when the symbol is placed. (The coordinates returned in the position field of the OVwSymbolInfo structure are in the virtual coordinate system of the submap.)

Coordinate placement can be used to position symbols relative to one another or at some fixed point on background graphics. See “OVwSetBackgroundGraphic(3)” on page 801. Submaps without background graphics are scaled so that unused space on the edges of the virtual coordinate system for the submap is not displayed. Unused space is not clipped in this way for submaps with background graphics, since symbol placement is maintained relative to the background graphics. The virtual size of a submap with background graphics is determined by the size of the background graphics, so that a symbol added at position (50, 50) on a (200 x 200) grid will appear at a position 25% of the way from the upper left-hand corner along both the X and Y axes. When the background graphics are scaled because of a change in the size of the submap window, the symbol will remain at the same position relative to the background graphics.

The grid size can be used to determine how large symbols appear on the submap. For example, two symbols placed at positions (25, 25) and (75, 75) on a (100 x 100) grid will appear in the same positions relative to one another as symbols placed at positions (250, 250) and (750, 750) on a (1000 x 1000) grid, but the symbols in the latter case will appear smaller because of the greater distance between them.

If the symbol is moved on a submap with a non-sequence layout, it will be placed at the appropriate coordinate position and have a placement style of ovwCoordPosition. If the submap has a sequence layout, the symbol will be placed at the coordinate position and then the virtual coordinate position will be used to determine a relative position in the sequence; the symbol will thereafter have a placement style of ovwSequencePosition. If the submap has a sequence layout and automatic layout is enabled, the automatic layout algorithm is executed to evenly position the symbols and the coordinate position will be lost.

If the symbol being moved has a current placement value of ovwStarCenterPosition, it will be moved to the new coordinate position, but it will have a placement value of ovwStarCenterPosition.

It is important to note that symbol positions specified by coordinates are lost whenever the automatic layout algorithm is executed. For the application to disallow all automatic layout, the submap should be created with the layout ovwNoLayout.

Sequence Position: If the placement is specified as ovwSequencePosition, the symbol is first removed from its previous position. If automatic layout is enabled for the submap, the symbol is then placed in the sequence immediately after the specified predecessor symbol and the layout algorithm is executed to evenly space all symbols. If automatic layout is disabled, the symbol is added to the symbol holding area; the predecessor information for the symbol is saved and used if the symbol is subsequently included in the submap using automatic layout. A value of ovwNullSymbolId for the predecessor symbol ID will result in the symbol being placed as the first symbol in the sequence.

Star Center Position: If the placement is specified as ovwStarCenterPosition, the symbol is removed from its previous position and placed as the center of a star layout. The setting of a symbol as the star center takes effect regardless of whether automatic layout is enabled or disabled. If there is an existing star center, that symbol is displaced as the star center. If an existing star center is displaced and automatic layout is enabled for the submap, the displaced star center is placed on the submap by executing the layout algorithm for the submap. If an existing star center is displaced and automatic layout is disabled, the displaced star center is added to the symbol holding area. Making a symbol the star center causes it to be placed in the center of the star layout; it does not have any effect on the connections in a star submap.

Hidden Symbols: Hidden symbols, which are symbols that exist on the submap but are not displayed, can be moved. Moves that result in a placement of ovwSequencePosition or ovwStarCenterPosition will take effect but will not become visible until the symbol is unhidden. Moves that merely result in a change in the coordinate position of a symbol will be silently ignored, since the symbol is not visible anyway. An automatic layout will not be executed as a result of the move of a hidden symbol, unless a displaced star center needs to be placed and automatic layout is enabled.

Move Event: The ovwConfirmMoveSymbol event is generated as a result of a symbol being moved using OVwSetSymbolPosition. See “OVwConfirmMoveSymbolCB(3)” on page 599 for more information. After OVwSetSymbolPosition is called, the placement of a symbol can be returned in the move event as ovwNoPosition if (1) the symbol was added to the symbol holding area because automatic layout was disabled or (2) the move did not become effective immediately because the submap was not displayed.

Parameters

map	Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
position	Specifies a pointer to a structure describing the symbol position.
symbolId	Specifies the ID of the symbol.

Return Values

If successful, OVwSetSymbolPosition returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwSetSymbolPosition sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_ICON_SYMBOL_BAD_COORDS] The x or y coordinate specified in the position argument has a value that is less than zero or greater than the width or height that sets the scale for the coordinates.

[OVw_ICON_SYMBOL_BAD_GRID] The width or height specified in the position argument for setting the scale for the x and y coordinates has a value less than or equal to zero.

[OVw_ICON_SYMBOL_PRED_NOT_FOUND] The symbol specified in the position argument as the predecessor of the symbol symbolId does not exist on the same submap as the symbol symbolId.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_MAP_READ_ONLY] The map is open with read-only permission.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_SUBMAP_INVALID_SYMBOL_PLACEMENT] The placement field of the position argument has a value that is not valid for the layout of the submap on which the symbol symbolId appears.

[OVw_SUBMAP_PERMISSION_DENIED] The symbol exists on the application plane of an exclusive submap created by another application.

[OVw_SYMBOL_NOT_FOUND] The symbol specified by symbolId does not exist on the open map.

[OVw_SYMBOL_WRONG_VARIETY] The symbol specified by symbolId does not have variety ovwIconSymbol. Position can only be set for icon symbols.

Implementation Specifics

OVwSetSymbolPosition supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolPosition, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwConfirmMoveSymbolCB(3)” on page 599.
- See “OVwCreateSymbol(3)” on page 623.
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwSetBackgroundGraphic(3)” on page 801.
- See “OVwApiIntro(5)” on page 560.

OVwSetSymbolStatusSource(3)

Purpose

Sets status source of a symbol

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolStatusSource(OVwMapInfo *map, OVwSymbolId symbolId,
      int statusSource);
```

Description

OVwSetSymbolStatusSource changes the status source of a symbol on the open map.

If a symbol has status source ovwObjectStatusSource, status can only be set for the symbol using OVwSetStatusOnObject or OVwSetStatusOnObjects. This status source should be used when it is appropriate for the status of all symbols of an object to be the same and to reflect the object status.

If a symbol has status source ovwCompoundStatusSource, status is determined by user configurable status propagation rules implemented by the graphical interface. The status of the symbol is determined by the status of the symbols in the child submap of the object represented by the symbol.

If a symbol has status source ovwSymbolStatusSource, status can only be set for the symbol using OVwSetStatusOnSymbol or OVwSetStatusOnSymbols, described in “OVwSetStatusOnObject(3)” on page 803. This status source, in conjunction with OVwSetStatusOnSymbol, should be used when it is appropriate for a symbol to have its status determined by context. Thus, the different symbols representing an object could have different status depending on where they appear on the map. This status source also allows an application to implement its own status propagation rules.

You can change the status source of a symbol through the graphical interface.

an application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

Parameters

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

statusSource Specifies the status source for the symbol. The permitted values are defined in <OV/ovw.h>:

ovwObjectStatusSource	The symbol gets its status from the status of the object.
ovwCompoundStatusSource	The symbol gets its status through propagation from the child submap of the object.
ovwSymbolStatusSource	The symbol has its status set explicitly.

symbolId Specifies the ID of the symbol.

Return Values

If successful, OVwSetSymbolStatusSource returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwSetSymbolStatusSource sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_MAP_NOT_OPEN]	The argument map does not specify an open map.
[OVw_MAP_READ_ONLY]	The map is open with read-only permission.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.
[OVw_SUBMAP_PERMISSION_DENIED]	The symbol exists on the application plane of an exclusive submap created by another application.
[OVw_SYMBOL_INVALID_STATUS_SOURCE]	The argument statusSource has a value that is not valid.
[OVw_SYMBOL_NOT_FOUND]	The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwSetSymbolStatusSource supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolStatusSource, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwSetStatusOnObject(3)” on page 803.
- See “OVwApiIntro(5)” on page 560.

OVwSetSymbolType(3)

Purpose

Sets symbol type of a symbol

Syntax

```
#include <OV/ovw.h>

int OVwSetSymbolType(OVwMapInfo *map, OVwSymbolId symbolId,
     OVwSymbolType symbolType, unsigned int flags);
```

Description

OVwSetSymbolType changes the symbol type of a symbol. The symbol type determines the visual appearance of the symbol. It can also be used to initialize capability field values on the underlying object.

Symbol types are created by registering them in a symbol type registration file. See “OVwRegIntro(5)” on page 769 for more information. A symbol type has two components: a symbol class and a symbol subclass. A symbol type is represented as a string of the form “<class>:<subclass>” (for example, “Computer:Workstation”).

A symbol type has a variety (icon or connection) that is determined by the variety of its symbol class as defined in the symbol type registration file. Only symbol types of the icon variety can be used with symbols of the ovwIconSymbol variety, and only symbol types of the connection variety can be used with symbols of the ovwConnSymbol variety. (Symbol types belonging to the Connection symbol class are used for connection symbols.)

An application is not permitted to set or clear interest in a symbol on the application plane of an exclusive submap created by another application.

Parameters

flags Specifies flags that can be used when setting the symbol type. These are the same flags available when creating a symbol. This is the logical OR of the following flags defined in <OV/ovw.h>:

ovwNoSymbolFlags	This value can be specified if no flags are needed.
ovwMergeDefaultCapabilities	The default capability field values for the symbol type symbolType will be set on the symbol’s object for those fields that do not already have values set. Default capabilities are defined in the symbol registration file.
ovwDoNotDisplayLabel	The symbol label will not be displayed. Leaving this flag cleared will not affect whether or not the symbol label is displayed.

map Specifies a pointer to the MapInfo structure for an open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbolId Specifies the ID of the symbol.

symbolType Specifies the symbol type to use for displaying the symbol. Symbol-type values are defined in the symbol type registration files. Some predefined symbol types are also listed in the <OV/sym_types.h> header file. For connection symbols, a NULL value can be used to indicate the default connection symbol type. A NULL value is not allowed for icon symbols.

Return Values

If successful, OVwSetSymbolType returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwSetSymbolType sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST] The connection to the NetView for AIX program was lost.

[OVw_MAP_NOT_OPEN] The argument map does not specify an open map.

[OVw_MAP_READ_ONLY] The map is open with read-only permission.

[OVw_OUT_OF_MEMORY] A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED] The EUI API has not been initialized with OVwInit.

[OVw_SUBMAP_PERMISSION_DENIED] The symbol exists on the application plane of an exclusive submap created by another application.

[OVw_SYMBOL_INVALID_FLAGS] The argument flags has a value that is not valid.

[OVw_SYMBOL_NOT_FOUND] The symbol specified by symbolId does not exist on the open map.

[OVw_SYMBOL_TYPE_NOT_FOUND] The argument symbolType is not a registered symbol type.

[OVw_SYMBOL_TYPE_WRONG_VARIETY] The argument symbolType has a symbol type variety (icon or connection) that does not match the variety of the symbol specified by symbolId.

Implementation Specifics

OVwSetSymbolType supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolType, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwGetMapInfo(3)” on page 719.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwShowHelp(3)

Purpose

Requests presentation of help information

Syntax

```
#include <OV/ovw.h>

int OVwShowHelp(unsigned long helpType, char *helpRequest);
```

Description

OVwShowHelp enables applications to implement an integrated, context-sensitive help system. It requests that the NetView for AIX help system display the specified help for the application. The type of request and request string are relayed by the NetView for AIX program to the designated NetView for AIX help application, ovhelp, which processes the help request appropriately.

Parameters

helpRequest
Specifies a pointer to the name of the help to be displayed. Generally, this string is a file path relative to the application's registered help directory. The string is processed according to the specified helpType.

helpType
Specifies the type of help being requested. The standard help types are defined in the OV/ovw.h header file:

ovwHelpIndex
The path of an ovhelp help index, relative to the registered help directory, is assumed to be helpRequest. If the application has not registered a help directory, the path is assumed to be relative to the NetView for AIX help directory, /usr/OV/help/$LANG.

ovwHelpFile
The path of an ovhelp help file, relative to the registered help directory, is assumed to be helpRequest. If the application has not registered a help directory, the path is assumed to be relative to the NetView for AIX help directory, /usr/OV/help/$LANG.

Return Values

If successful, OVwShowHelp returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwShowHelp sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_APP_NOT_FOUND]	The NetView for AIX help system application, ovhelp, is not running.
[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Implementation Specifics

OVwShowHelp supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwShowHelp, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovhelp(1).
- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

OVwSubmapCloseCB(3)

Purpose

Functions as a callback for a submap-close event

Syntax

```
#include <OV/ovw.h>

void (*OVwSubmapCloseCB) (void *userData, OVwEventType type,
      OVwMapInfo *map, OVwSubmapInfo *submap);
```

Description

To receive an event indicating that a submap is being closed, use OVwAddCallback to register a callback function of type OVwSubmapCloseCB, to be called when an ovwSubmapClose event is generated. You can close a submap through the graphical interface.

A map-close event is generated when a map is closed through the graphical interface. (See "OVwMapCloseCB(3)" on page 759.) A map close event implies that the submaps within the map were closed. OVwSubmapCloseCB is not generated when a submap is closed because a map is closed.

Parameters

map	Specifies a pointer to the OVwMapInfo structure for the open map.
submap	Specifies a pointer to the OVwSubmapInfo structure for the submap being closed.
type	Specifies the type of event that caused this callback to be invoked, namely ovwSubmapClose.
userData	Specifies a pointer to the user data registered for the callback.

Implementation Specifics

OVwSubmapCloseCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSubmapCloseCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwSubmapOpenCB(3)” on page 829.
- See “OVwApiIntro(5)” on page 560.
- See “OVwMapCloseCB(3)” on page 759.

OVwSubmapOpenCB(3)

Purpose

Functions as a callback for a submap-open event

Syntax

```
#include <OV/ovw.h>

void (*OVwSubmapOpenCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSubmapInfo *submap);
```

Description

To receive an event indicating that a submap is being opened, use OVwAddCallback to register a callback function of type OVwSubmapOpenCB, to be called when an ovwSubmapOpen event is generated. A submap-open event is generated when a submap is displayed with the OVwDisplaySubmap routine or opened by a user through the graphical interface. The ovwSubmapOpen event occurs for the home submap at startup before registered applications have been started, so applications do not receive this initial submap open event.

Parameters

map	Specifies a pointer to the OVwMapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
submap	Specifies a pointer to the OVwSubmapInfo structure for the submap being opened.
type	Specifies the type of event that caused this callback to be invoked, namely ovwSubmapOpen.
userData	Specifies a pointer to the user data registered for this callback.

Implementation Specifics

OVwSubmapOpenCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwSubmapOpenCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwDisplaySubmap(3)” on page 683.
- See “OVwSubmapCloseCB(3)” on page 827.
- See “OVwApiIntro(5)” on page 560.

OVwUserSubmapCreateCB(3)

Purpose

Functions as a callback for a user, submap-create event

Syntax

```
#include <OV/ovw.h>

void (*OVwUserSubmapCreateCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolInfo *symbol, OVwSubmapInfo *submap)
```

Description

To receive an event indicating that a user is attempting to open a submap that has not been created yet, use OVwAddCallback to register a callback function of type OVwUserSubmapCreateCB, to be called when an ovwUserSubmapCreate event is generated. This is useful for applications that need to create submaps on user request. An ovwUserSubmapCreate event is generated when a user tries to open a submap that does not yet exist by double-clicking on an explodable symbol that has an object without a child submap. An application will receive the OVwUserSubmapCreate callback only if it has registered application interest in the symbol on which the user double-clicked and that symbol does not have a submap. See "OVwSetSymbolApp(3)" on page 808 for more information.

The routine OVwAckUserSubmapCreate should always be called in the callback for the ovwUserSubmapCreate event to indicate whether the application had created a submap in response to the user action. If the application does not create a submap, the user will be prompted and given the opportunity to create a submap.

Parameters

map	Specifies a pointer to the OVwMapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
submap	Specifies a pointer to the OVwSubmapInfo structure for the submap on which the symbol exists that the user is trying to open.
symbol	Specifies a pointer to the OVwSymbolInfo structure for the symbol that the user is trying to open to display its child submap.
type	Specifies the type of event that caused this callback to be invoked, namely ovwUserSubmapCreate.
userData	Specifies a pointer to the user data registered for this callback.

Examples

The following code fragment illustrates how to register a callback routine for receiving a user, submap-create event:

```
void userSubmapCreateCB(void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolInfo *symbol,
       OVwSubmapInfo *submap)
       {
                OVwSubmapId submap_id = ovwNullSubmapId;

                /*
                 * Check whether it is appropriate
                 for the application
                 * to create a submap.  If so, create
                 submap and set
                 * submap_id.
                 */

                OVwAckUserSubmapCreate(map, submap_id,
                symbol->symbol_id);
        }

        OVwAddCallback(ovwUserSubmapCreate, NULL,
              (OVwCallbackProc) userSubmapCreateCB, NULL);
```

Implementation Specifics

OVwUserSubmapCreateCB supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwUserSubmapCreateCB, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAckUserSubmapCreate(3)” on page 530.
- See “OVwAddCallback(3)” on page 539.
- See “OVwApiIntro(5)” on page 560.

OVwVerifyAdd(3)

Purpose

Validates the initial description information for an object when a user adds its symbol to the open map

Related Functions

OVwQueryAddSymbolCB
OVwConfirmAddSymbolCB

Syntax

```
#include <OV/ovw.h>

int OVwVerifyAdd(OVwMapInfo *map, OVwFieldBindList *dialogBoxFields,
        OVwBoolean verified, OVwBoolean appPlane, char *errorMsg);

void (*OVwQueryAddSymbolCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwSubmapInfo *submap,
        OVwFieldBindList *capabilityFields,
        OVwFieldBindList *dialogBoxFields);

void (*OVwConfirmAddSymbolCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwSymbolInfo *symbol,
        OVwFieldBindList *capabilityFields,
        OVwFieldBindList *dialogBoxFields);
```

Description

These callbacks manage events sent to applications that have registered to receive them whenever an add operation is selected by the user or requested by another application. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API, including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when an object is added should register these callbacks through the OVwAddCallback function call, using ovwQueryAddSymbol and ovwConfirmAddSymbol as the callback types.

When the user adds a symbol to the map, an add dialog box is presented on the display. If the application has enrolled fields for the add dialog box through the Application Registration File, the user has the option to open an application specific add-dialog box.

If the user opens the application-specific add-dialog box, there are several buttons on the bottom of that box: OK, Verify, and Cancel. When this dialog box is first opened, the OK button is grayed out and cannot be pressed, but the Verify and Cancel buttons can be used. After filling in the dialog box fields, the user should press the Verify button. The NetView for AIX program will send an OVwQueryAddSymbol event to the appropriate application. The NetView for AIX program will include a list of all fields enrolled in that dialog box and their values in the dialogBoxFields parameter. All of the capability fields and their values will be sent in the capabilityFields parameter.

Upon receipt of this event, the callback routine must determine whether the user-specified information is valid and whether the symbol should be added to the application or the user plane. If so, the application must call OVwVerifyAdd with the verified parameter set to TRUE. If any of the user-specified information is not valid or the symbol should not be added to the application or user plane, the application must call

OVwVerifyAdd with the verified parameter set to FALSE. If the application fails to call OVwVerifyAdd in response to an ovwQueryAddSymbol event, the graphical interface will hang.

The field values entered by the user are valid if they are complete and consistent with the model being enforced by the application. For example, for an application enforcing IP rules, the IP Address field has very strict rules about what an IP address can look like. The value “15.two.100.three” is an incorrect value because alpha characters are not allowed.

When testing for correctness, the application must also consider the submap to which the symbol is being added. The submap will likely play a significant part in determining if the information is valid. For example, an IP application can enforce the rule that a submap contains symbols for objects that are connected to a particular network. If the user attempts to add a symbol for an object that is not connected to that network, then the information is not valid (although it can be valid without the context of the submap).

There can be fields for which the application cannot determine correctness. A field that is strictly for user convenience, such as comments, will not be tested by the application. Any information entered in these fields is assumed correct. Therefore only a subset of the fields in the dialog box will actually contribute to the validity of the new object.

The application and user planes are concepts used by the NetView for AIX program to separate symbols based on the validity of their information. Objects whose symbols are on the application plane in a particular submap will all have valid information in the context of that submap. An application is not permitted to place the symbol of an object that has information that is not valid onto the application plane. The user plane is for symbols of objects that are, for whatever reason, not valid in the context of the submap.

The application plane is typically used to represent objects that are semantically correct, whereas the user plane can contain anything. For example, an application might set the status of symbols only on the application plane. Most applications will not acknowledge the existence of symbols on the user plane.

If all input fields are valid, the callback routine responds by calling OVwVerifyAdd with the verified field set to TRUE and the appPlane field set to TRUE. If the information is valid but not for the submap specified, the verified field is set to TRUE and the appPlane field is set to FALSE so the symbol is placed in the user plane. If there is a problem, then the application calls OVwVerifyAdd with the verified field set to FALSE and the appPlane field set to FALSE. In this case, the appPlane field cannot be set to TRUE. If either field is set to FALSE then an error message should be passed.

In OVwVerifyAdd, the dialogBoxFields parameter is a list of fields from the dialogBoxFields list passed by OVwQueryAddSymbolCB. The values of these fields can be left unchanged or a modified version of them can be returned. The dialog box will be updated to reflect the values returned by the application.

When verified is TRUE, the OK button on the dialog box is made operable so the user can press it. Once the OK button is pressed on the top level add dialog box (not the application-specific dialog box), the NetView for AIX program will send another event, OVwConfirmAddSymbol, to the application. When this event is received, the callback routine processes the add.

If, after pressing the Verify button, the user changes one of the fields, the OK button is again grayed out by the graphical interface. Now the user must re-verify the information on the screen before being allowed to press OK.

The OVwConfirmAddSymbol events can also be sent to an application if another application adds a symbol to a submap. In this case, the OVwQueryAddSymbol event is not sent. As a result, the information in dialogBoxFields might not be valid because the application that created the symbol might not check what the values are. This means that an application cannot depend on the dialogBoxFields values to be correct and therefore must recheck them when the OVwConfirmAddSymbol event is received. If one or

more of the values is found to be incorrect, then the application can assume that this symbol is being added by another application and can be ignored.

InitialVerify and ImmediateVerify are two of the options that can be used in the registration file when enrolling fields in the dialog box. See “OVwRegIntro(5)” on page 769 for a more detailed description. These options are useful for making the dialog box interactions easier to use. They are similar; each causes the NetView for AIX program to send an ovwQueryAddSymbol event before the user presses the Verify button. The difference is the action that triggers the NetView for AIX program to send the event. If a dialog box has the InitialVerify option set, the NetView for AIX program will send the ovwQueryAddSymbol event when the user opens the application-specific describe dialog box but before it is actually displayed. It enables the application to fill in some default values for the fields, by setting the values of the fields in dialogBoxFields and calling OVwVerifyAdd. After the application makes this call, the NetView for AIX program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, the NetView for AIX program sends the ovwQueryAddSymbol event whenever the user exits that field. This can be used by the application to fill in other fields based on the value of this field as a convenience to the user.

Parameters

appPlane	Specifies whether the object can be added to the application plane of the submap identified by submap ID. If the value of this field is TRUE, the symbol is allowed on the application plane. If not, the object will be added to the user plane. The appPlane field should be FALSE is the verified field is FALSE.
capabilityFields	Specifies a pointer to a list of the capability fields set for this object along with their values.
dialogBoxFields	Specifies a pointer to a list of the fields in the Add dialog box along with their values.
errorMsg	Specifies error message to display to the user. This is a NULL-terminated string that the graphical interface displays in the messages field in the add dialog box. The string is auto-wrapped to match the width of the messages field. You can use newline characters (ln) to force line endings if you want to have some formatting control.
map	Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
submap	Specifies a pointer to the SubmapInfo structure for the submap to which the object is being added.
symbol	Specifies a pointer to the SymbolInfo structure for the symbol added to the current map.
type	Specifies the type of the event that caused OVwQueryAddSymbolCB or OVwConfirmAddSymbolCB to be invoked, namely ovwQueryAddSymbol or ovwConfirmAddSymbol. This is useful for callbacks that handle several event types.
userData	Specifies a pointer to the user data registered for the callback.
verified	Specifies whether field information is consistent and complete according to the application. If verified is TRUE, the user can make the changes; otherwise, the user cannot make them.

Return Values

If successful, OVwVerifyAdd returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwVerifyAdd sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

- The following example illustrates a typical add interaction between the NetView for AIX program and an application.

```
void
queryAddProc(void *userData, OVwEventType type, OVwMapInfo *map,
             OVwSubmapInfo *submap,
             OVwFieldBindList *capabilityFields,
             OVwFieldBindList *dialogBoxFields);
{
    /* verify the field values in dialogBoxFields */

    if (infoIsOK()) {
    /* say OK and put on appPlane */
        if ( OVwVerifyAdd (map,
                           dialogBoxFields,
                           TRUE, TRUE, NULL)); {
            /* bail out because of error in OVwVerifyAdd */
        }
    } else {
    /* say not OK and why */
        if ( OVwVerifyAdd (map,
                           dialogBoxFields,
                           FALSE, FALSE, "bogus data"))
            /* bail out because of error in OVwVerifyAdd */
        }
    }
}

void
confirmAddProc(void *userData, OVwEventType type, OVwMapInfo *map,
            OVwSubmapInfo *submap, OVwSymbolInfo *symbol,
            OVwFieldBindList *capabilityFields);
            OVwFieldBindList *dialogBoxFields,
{
    /* recheck the dialogBoxFields here since
    this event may be sent to us as a result of an add by
    another application. */

    /* if the field values are OK, add the
    symbol to our internal structure. */
}
```

Implementation Specifics

OVwVerifyAdd and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwVerifyAdd or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwCreateSubmap(3)” on page 619.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwVerifyAppConfigChange(3)

Purpose

Validates the user change of application configuration values

Related Functions

OVwQueryAppConfigCB
OVwConfirmAppConfigCB

Syntax

```
#include <OV/ovw.h>

int OVwVerifyAppConfigChange(OVwMapInfo *map,
        OVwFieldBindList *configParams,
        OVwBoolean verified, char *errorMsg);

void (*OVwQueryAppConfigCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwFieldBindList *configParams);

void (*OVwConfirmAppConfigCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwFieldBindList *configParams);
```

Description

The callbacks OVwQueryAppConfigCB and OVwConfirmAppConfigCB handle events sent to applications that have registered to receive them when the user changes the application configuration. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API, including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when the configuration is changed should register these callbacks through the OVwAddCallback function call, using ovwQueryAppConfigChange and ovwConfirmAppConfigChange, respectively, as the event types.

An ovwQueryAppConfigChange event is sent when the user modifies the application configuration parameters for the currently open map. To do this, the user can use the Configuration dialog box specific to this application. The Application Registration File is used to enroll fields in this dialog box; see “OVwRegIntro(5)” on page 769 for more information about this file.

When the application's Configuration dialog box is presented, it has several buttons on the bottom of the box: OK, Verify, and Cancel. When you type anything in the box, the OK button is grayed out so it cannot be pressed. The Verify and Cancel buttons can be used. When you modify any field, you should press the Verify button so the application can verify the changes. The NetView for AIX program will send an ovwQueryAppConfigChange event to the application.

The OVwQueryAppConfigCB routine is invoked upon receipt of this event. This routine must determine whether the field values entered by the user are valid. The definition of valid is completely determined by the application. An example of a field with an invalid value is an integer field containing a value that is specified out of range.

Note: When the user selects `Edit..New Map`, a NULL map pointer is passed to the OVwQueryAppConfigCB callback routine, and that NULL pointer can be passed to the

OVwVerifyAppConfigChange function. No ovwConfirmAppConfigChange event will be generated in this situation.

All fields in the dialog box, with their values, are in the configParams parameter. The callback routine can update these values by changing the values in the list and passing a pointer to the list in the OVwVerifyAppConfigChange call. If no change is necessary, the list is passed back unchanged.

If the callback routine determines that the field values are valid, it responds by calling OVwVerifyAppConfigChange with the verified flag set to TRUE. The graphical interface will then make the OK button operable so the user can commit the changes. If there was a problem found with one or more of the fields, the callback routine calls OVwVerifyAppConfigChange with the verified field set to FALSE and errorMsg set to an appropriate message. In this case, the graphical interface will keep the OK button disabled and display the message so that the user can fix the problem and re-verify the information before being permitted to commit the data.

After the data in the Configuration dialog box has been verified and the user presses the OK button, the NetView for AIX program sends an ovwConfirmAppConfigChange event to the application. The OVwConfirmAppConfigCB handles this event.

InitialVerify and ImmediateVerify are two of the options that can be used in the registration file when enrolling fields in the dialog box. See “OVwRegIntro(5)” on page 769 for a more detailed description. These options are useful for making the dialog box interactions easier to use. They are similar; each causes the NetView for AIX program to send an ovwQueryAppConfig event before the user presses the Verify button. The difference is the action that triggers the NetView for AIX program to send the event. If a dialog box has the InitialVerify option set, the NetView for AIX program will send the ovwQueryAppConfigChange event when the user opens the application-specific describe dialog box but before it is actually displayed. It enables the application to fill in some default values for the fields, by setting the values of the fields in ConfigParams dialogBoxFields and calling OVwVerifyAppConfigChange. After the application makes this call, the NetView for AIX program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, then the NetView for AIX program sends the ovwQueryAppConfigChange event when the user exits that field. This can be used by the application to fill in other fields, based on the value of this field, as a convenience to the user.

Parameters

configParams	Specifies a pointer to application-specific configuration parameters for the current map.
errorMsg	Specifies a pointer to error message to display to the user. It is a NULL-terminated string that the graphical interface will display in the messages field of the dialog box. The string is auto-wrapped by the graphical interface to fit the width of the messages field. You can use a newline character (ln) to force a line ending if you want some formatting control.
map	Specifies a pointer to the MapInfo structure for the currently open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
type	Specifies the type of event which caused this callback routine to be invoked, namely ovwqueryAppConfig or ovwConfirmAppConfig. This is useful if a single callback handles multiple events types.

verified	Specifies whether field information is consistent and complete according to the application. If verified is TRUE, the user can commit the changes; otherwise, the user cannot commit them.
userData	Specifies a pointer to the user data registered for the callback.

Return Values

If successful, OVwVerifyAppConfigChange returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwVerifyAppConfigChange sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

See “OVwVerifyAdd(3)” on page 833 for an example interaction. The interaction model for OVwVerifyAppConfigChange is similar.

Implementation Specifics

OVwVerifyAppConfigChange and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwVerifyAppConfigChange or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwError(3)” on page 688.
- See “OVwGetAppConfigValues(3)” on page 698.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwVerifyConnect(3)

Purpose

Validates the user-selected connect operation for two symbols

Related Functions

OVwQueryConnectSymbolsCB
OVwConfirmConnectSymbolsCB

Syntax

```
#include <OV/ovw.h>

int OVwVerifyConnect(OVwMapInfo *map, OVwObjectInfo *object1,
       OVwObjectInfo *object2, OVwFieldBindList *dialogBoxFields,
       OVwBoolean verified, OVwBoolean appPlane,
       char *errorMsg);

void (*OVwQueryConnectSymbolsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSubmapInfo *submap, OVwObjectInfo *object1,
       OVwObjectInfo *object2, OVwFieldBindList *capabilityFields,
       OVwFieldBindList *dialogBoxFields);

void (*OVwConfirmConnectSymbolsCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwSymbolInfo *symbol, OVwObjectInfo *object1,
       OVwObjectInfo *object2, OVwFieldBindList *capabilityFields,
       OVwFieldBindList *dialogBoxFields);
```

Description

The callbacks OVwQueryConnectSymbolsCB and OVwConfirmConnectSymbolsCB handle events sent to applications that have registered to receive them when the user selects the connect operation. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API, including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when the connect operation is selected by the user or another application should register these callbacks by using the OVwAddCallback function call and by using ovwQueryConnectSymbols and ovwConfirmConnectSymbols as the event types.

When you add a connection between two symbols on the map, the graphical interface displays an add connection dialog box. If your application has enrolled fields for the add connection dialog box, you can open an add connection dialog box specific to your application. See “OVwRegIntro(5)” on page 769 for more information.

If you open the application-specific dialog box, there are several buttons on the bottom of that box: OK, Verify, and Cancel. When this dialog box is first opened, the OK button is grayed out and cannot be pressed but the Verify and Cancel can be used. When you have filled in the dialog box fields, you should press the Verify button. The NetView for AIX program will send an ovwQueryConnectSymbols event to the appropriate application. The NetView for AIX program will include a list of all fields enrolled in that dialog box and their values in the dialogBoxFields parameter. All of the capability fields and their values will be sent in the capabilityFields parameter.

Upon receipt of this event, the callback routine must determine whether the field values are correct. If they are, the application must call OVwVerifyConnect with the verified parameter set to TRUE. If any of the fields are not correct, the application must call OVwVerifyConnect with the verified parameter set to FALSE. If the application fails to call OVwVerifyConnect in response to an ovwQueryConnectSymbols event then the graphical interface will hang.

If OVwVerifyConnect is called with the verified parameter set to TRUE, then the graphical interface will make the OK button operable on the add connection dialog box. When this occurs, you can press that button to continue with the operation. After you press the OK button on the application-specific dialog box, the main add connection dialog box is still open. When the OK button on the main connect dialog box is pressed, the NetView for AIX program sends an ovwConfirmConnectSymbols event to the appropriate applications.

When the ovwConfirmConnectSymbols is received, the application must make sure the dialogBoxFields are correct, even though they were already checked in OVwQueryConnectSymbolsCB. This is required because the ovwConfirmConnectSymbols event can be sent to an application as a result of another application's adding a connection to a submap. In this case, the ovwQueryConnectSymbols event is not generated. If the dialogBoxFields are found to be correct, you can make a record of this information and proceed with processing. If any of the dialogBoxFields are found to be incorrect, your application can assume that the connection is being added by another application and can be ignored.

InitialVerify and ImmediateVerify are two of the options that you can use in the registration file when enrolling fields in the dialog box. See “OVwRegIntro(5)” on page 769 for a more detailed description. These options are useful for making the dialog box interactions easier to use. They are similar; each causes the NetView for AIX program to send an ovwQueryConnectSymbols event before the user presses the Verify button. The difference is the action that triggers the NetView for AIX program to send the event. If a dialog box has the InitialVerify option set, the NetView for AIX program will send the ovwQueryConnectSymbols event when the user opens the application-specific describe dialog box but before it is actually displayed. It enables the application to fill in some default values for the fields, by setting the values of the fields in dialogBoxFields and calling OVwVerifyConnect. After the application makes this call, the NetView for AIX program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, the NetView for AIX program sends the ovwQueryConnectSymbols event when the user exits that field. This can be used by the application to fill in other fields based on the value of this field as a convenience to the user.

Parameters

capabilityFields	Specifies a pointer to a list of capability fields set for the connect object being added.
dialogBoxFields	Specifies a pointer to a list of dialog box fields.
errorMsg	Specifies a pointer to error message to display to the user. This is a NULL-terminated string that the graphical interface will display in the messages field of the add connection dialog box. The string is auto-wrapped to fit the width of the field. You can use a newline character (ln) to force a line ending if you want some formatting control.
map	Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.
object1	Specifies a pointer to the ObjectInfo structure for object1.

object2	Specifies a pointer to the ObjectInfo structure for object2.
appPlane	Specifies whether the object can be added to the application plane of the submap identified by submap ID. If not, the object will be added to the user plane. The appPlane field should be FALSE if the verified field is FALSE.
submap	Specifies a pointer to the SubmapInfo structure for the submap to which the symbol is being added.
symbol	Specifies a pointer to the SymbolInfo structure for the symbol added to the current map.
type	Specifies the type of event that caused the callback to be invoked, namely ovwQueryConnectSymbols or ovwConfirmConnectSymbols. This is useful for callbacks that handle several event types.
userData	Specifies a pointer to the user data registered for the callback.
verified	Specifies whether field information is consistent and complete according to the application. If verified is TRUE, the user will be allowed to commit the changes; otherwise, no changes are allowed.

Return Values

If successful, OVwVerifyConnect returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwVerifyConnect sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

See the example for OVwQueryAddSymbol in “OVwVerifyAdd(3)” on page 833 for a sample interaction. The interaction model for OVwQueryAddSymbolCB is similar to that for OVwQueryConnectSymbolsCB.

Implementation Specifics

OVwVerifyConnect supports single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwVerifyConnect and its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwVerifyDeleteSymbol(3)

Purpose

Validates the deletion of symbols

Related Functions

OVwQueryDeleteSymbolsCB
OVwConfirmDeleteSymbolsCB

Syntax

```
#include <OV/ovw.h>

int OVwVerifyDeleteSymbol(OVwMapInfo *map,
        OVwSymbolVerifyList *symbolVerifyList);

void (*OVwQueryDeleteSymbolsCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwSymbolVerifyList *symbolVerifyList);

void (*OVwConfirmDeleteSymbolsCB) (void *userData, OVwEventType type,
        OVwMapInfo *map, OVwSymbolList *symbolList);
```

Description

The callbacks handle events sent to applications that have registered to receive them whenever a delete operation is performed. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when a delete operation is selected by the user or another application should register these callbacks through the OVwAddCallback function call, using ovwQueryDeleteSymbols and ovwConfirmDeleteSymbols as the callback types.

Delete events are generated as a result of the user or another application deleting symbols or objects from the map. When the user deletes a symbol from the map, the NetView for AIX program will send a ovwQueryDeleteSymbols event to the applications that have registered interest in that symbol. Applications can receive OVwQueryDeleteSymbolsCB only if they have registered interest in the symbol that has been deleted. The application can allow or deny each individual symbol delete operation. If the application determines that the operation is valid then it sets the verified parameter for that symbol to TRUE, otherwise verified is set to FALSE. Once all symbols in the list are processed, the application must call OVwVerifyDeleteSymbol sending the list of symbols back to the NetView for AIX program. Once the NetView for AIX program gets the list back, it will selectively delete the symbols. The symbols for which the application set verified to TRUE are deleted while the symbol for which verified was FALSE are not. A dialog box for the symbols that were not deleted is then displayed that tells the user that the delete operation was denied and that the user can hide the symbol.

In the case where an application is deleting symbols from the map, there are no ovwQueryDeleteSymbols events generated only ovwConfirmDeleteSymbols. Applications should, therefore, be very careful about what symbols they delete from shared submaps since other applications may depend on their existence.

For the set of deleted symbols, the NetView for AIX program sends a ovwConfirmDeleteSymbols event. When the application receives the ovwConfirmDeleteSymbols event it must delete that symbol from its internal structures since it has been removed from the map.

If the last symbol for a given object gets deleted then the graphical interface deletes that object and sends an ovwConfirmDeleteObject event. See “OVwConfirmDeleteObjectsCB(3)” on page 588 for more details.

Parameters

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

symbolVerifyList
Specifies a pointer to a list of SymbolVerify structures that represent each symbol to be deleted. Included in the structure is a flag that indicates if that symbol can be deleted. If not, the graphical interface will tell the user and allow him to hide the object.

symbolList
Specifies a pointer to a list of SymbolInfo structures for the symbols to be deleted.

type
The type of event that caused the callback to be invoked, namely ovwQueryDeleteSymbols or ovwConfirmDeleteSymbols. This is useful if one callback handles multiple event types.

userData
Specifies a pointer to the user data registered for the callback.

Return Values

If successful, OVwVerifyDeleteSymbol returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwVerifyDeleteSymbol sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

See the example in “OVwVerifyAdd(3)” on page 833 for a sample interaction.

Implementation Specifics

OVwVerifyDeleteSymbol and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwVerifyDeleteSymbol or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwConfirmDeleteObjectsCB(3)” on page 588.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwVerifyDescribeChange(3)

Purpose

Validates the user change of description information for an object

Related Functions

OVwQueryDescribeCB
OVwConfirmDescribeCB

Syntax

```
#include <OV/ovw.h>

int OVwVerifyDescribeChange(OVwMapInfo *map, OVwObjectInfo *object,
       OVwFieldBindList *dialogBoxFields, OVwBoolean verified,
       char *errorMsg);

void (*OVwQueryDescribeCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectInfo *object,
       OVwFieldBindList *dialogBoxFields);

void (*OVwConfirmDescribeCB) (void *userData, OVwEventType type,
       OVwMapInfo *map, OVwObjectInfo *object,
       OVwFieldBindList *dialogBoxFields);
```

Description

The callbacks handle events sent to applications that have registered to receive them whenever a describe operation is selected by the user. See “OVwApiIntro(5)” on page 560 for an overview of the EUI API including the role of the asynchronous NetView for AIX events.

An application that needs to be notified when a describe operation is selected should register these callbacks through the OVwAddCallback function call, using ovwQueryDescribe and ovwConfirmDescribe as the event types.

An ovwVerifyDescribeChange event is sent when the user modifies the object describe parameters for the currently open map. To do this, you can use the Object Description dialog box specific to this application. The Application Registration File is used to enroll fields in this dialog box; see “OVwRegIntro(5)” on page 769 for more information about this file.

When the application's Object Description dialog box is presented, it has several buttons on the bottom of that box: OK, Verify, and Cancel. When the box is first opened, the OK button is grayed out so it cannot be pressed. The Verify and Cancel buttons remain operable. When you modify any field, you are expected to press the Verify button so the application can verify the changes. When the Verify button is pressed, the NetView for AIX program will send an ovwQueryDescribe event to the application. One of the parameters of this event is dialogBoxFields which contains a list of the fields enrolled in this dialog box along with their values.

Upon receipt of this event, it is the responsibility of the application to determine if the values of these fields are correct. If they are then the application must call OVwVerifyDescribeChange with the verified parameter set to TRUE. If any of the fields are not correct then it must call OVwVerifyDescribeChange with the

verified parameter set to FALSE. If the application fails to call OVwVerifyDescribeChange then the graphical interface will hang.

If OVwVerifyDescribeChange is called with the verified parameter set to TRUE, then the graphical interface will make the OK button operable on the Object Description dialog box. Then the user can press that button to continue with the operation. After pressing the OK button on the application-specific dialog box, the main Object Description dialog box is still open. When the OK button on the main Object Description dialog box is pressed, the NetView for AIX program sends an ovwConfirmDescribeChange event to the appropriate applications.

When the ovwConfirmDescribe is received, the application must make the update to its internal structures and/or its database since the NetView for AIX object database has been changed to reflect these new values. Since the dialogBoxFields were already checked for correctness when the ovwQueryDescribe was sent, the OVwConfirmDescribeCB routine need not recheck them.

Two of the options that can be used in the registration file when enrolling fields in the dialog box are InitialVerify and ImmediateVerify. See “OVwRegIntro(5)” on page 769 for a more detailed description. These options are very useful for making the dialog box interactions easier to use. They are very similar in nature. They each cause the NetView for AIX program to send an ovwQueryDescribe event before the user presses the Verify button. The difference is the action that triggers the NetView for AIX program to send the event. If a dialog box has the InitialVerify option set then the NetView for AIX program will send the ovwQueryDescribe event when the user opens the application-specific Object Description dialog box but before it is actually displayed. It gives the application the chance to fill in some default values for the fields. This is accomplished by setting the values of the fields in dialogBoxFields and calling OVwVerifyDescribeChange. After the application makes this call, the NetView for AIX program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, then the NetView for AIX program sends the ovwQueryDescribe event whenever the user exits that field. This can be used by the application to fill in other fields based on the value of this field as a convenience to the user.

Parameters

dialogBoxFields
Specifies a pointer to a list of application-specific add dialog box fields.

errorMsg
Specifies a pointer to error message to display to the user.

map
Specifies a pointer to the MapInfo structure for the open map. The map parameter can be obtained using OVwGetMapInfo or saved from the ovwMapOpen event using OVwCopyMapInfo.

object
Specifies a pointer to the ObjectInfo structure for the object for which the describe box semantic information is being modified.

type
The type of event that caused the callback routine to be invoked, namely ovwQueryDescribe or ovwConfirmDescribe. This is useful if one callback handles multiple event types.

verified
Specifies whether field information is consistent and complete according to the application. If verified is TRUE, the user will be allowed to commit the changes; otherwise, no changes are allowed.

userData
Specifies a pointer to the user data provided when the callback was added.

Return Values

If successful, OVwVerifyDescribeChange returns 0 (zero). If unsuccessful, it returns –1 (negative one).

Error Codes

OVwVerifyDescribeChange sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]	The connection to the NetView for AIX program was lost.
[OVw_OUT_OF_MEMORY]	A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]	The EUI API has not been initialized with OVwInit.

Examples

See the example in “OVwVerifyAdd(3)” on page 833 for a sample interaction. The interaction model for OVwVerifyAdd is similar to that for OVwVerifyDescribeChange.

Implementation Specifics

OVwVerifyDescribeChange and its related functions support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwVerifyDescribeChange or one of its related functions, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwAddCallback(3)” on page 539.
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.
- See “OVwRegIntro(5)” on page 769.

OVwXtAddInput(3)

Purpose

Registers the NetView for AIX event source with X

Related Functions

OVwXtAppAddInput

Syntax

```
#include <OV/ovw.h>

void *OVwXtAddInput();

void *OVwXtAppAddInput(void *app_c);
```

Description

OVwXtAddInput registers the NetView for AIX event source with X so that NetView for AIX events are processed during XtMainLoop or XtAppMainLoop.

OVwXtAppAddInput works like OVwXtAddInput, but for a specific X application context.

OVwXtAddInput and OVwXtAppAddInput should be used to allow X applications to use the EUI API. They should be called only after a successful call to OVwInit.

Note: You cannot use these calls with OVwXtMainLoop or OVwXtAppMainLoop. Choose the routine that works best for your X application.

Parameters

app_c
Specifies a pointer to the X application context for the application.

Return Values

OVwXtAddInput and OVwXtAppAddInput return, a void pointer to the XtInputId resulting from a call to XtAddInput or XtAppAddInput.

Examples

The following example shows the code for OVwXtAddInput. (Customize it as you choose.)

```
#include <X11/Intrinsic.h>
#include <OV/ovw.h>
#include <stdlib.>
#include <sys/stat.h>

/* An X input handler to dispatch NetView for AIX events */

void
__OVwXtInputCB(XtPointer closure, int *fd, XtInputId * id)
{
     struct stat buf;

     /*
     ** Flush any pending NetView for AIX events
     */
     while (OVwPending()) }
         if (OVwProcessEvent() < 0) {
             if (id && *id);
                 XtRemoveInput(*id);
             OVwDone();
             return;
         }
         /*
         **  If the file descriptor has gone bad, remove it from
         ** the Xt select loop.
         */
         if (fd && *fd >= 0) {
             if (fstat(*fd, &buf) < 0) {
                 if (id && *id)
                     XtRemoveInput(*id);
                 OVwDone();
                 return;
             }
         }
     }
}

void * /* XtInputId */
OVwXtAddInput()
{
     XtInputId id;
     int fd;

    fd = OVwFileDescriptor();
    if (fd >= 0)
        id = XtAddInput (fd, (XtPointer)XtInputReadMask, __OVwXtInputCB, NULL);
    return (void *)id;
}
```

Implementation Specifics

OVwXtAddInput and OVwXtAppAddInput support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwXtAddInput or OVwXtAppAddInput, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwXtMainLoop(3)” on page 855.
- See “OVwApiIntro(5)” on page 560.

OVwXtMainLoop(3)

Purpose

Dispatches X and NetView for AIX events continuously

Related Functions

OVwXtAppMainLoop

Syntax

```
#include <OV/ovw.h>
#include <X11/Intrinsic.h>

void OVwXtMainLoop();

void OVwXtAppMainLoop(void *app_c);
```

Description

OVwXtMainLoop is a replacement for XtMainLoop that dispatches NetView for AIX events and input events registered with the NetView for AIX program, in addition to X events. It is intended for use with applications that are using the Xt intrinsics.

OVwXtAppMainLoop is a replacement for XtAppMainLoop. It works like OVwXtMainLoop, but for a specific X application context.

Note: You cannot use these calls with OVwXtAddInput or OVwXtAppAddInput. Choose the routine that works best for your X application.

Parameters

app_c
Specifies a pointer to the X application context for the application.

Return Values

OVwXtMainLoop and OVwXtAppMainLoop have no return value.

Examples

The following example depicts the code for OVwXtMainLoop. (Customize it you choose.):

```
#include <X11/Intrinsic.h>
#include <OV/ovw.h>
#include <stdlib.h>
#include <sys/stat.h>

/* An X input handler to dispatch an NetView for AIX event */
void
__OVwInputCB(XtPointer closure, int *fd, XtInputId *id)
{
     struct stat buf;

     /* Process an NetView for AIX event */
     if (OVwProcessEvent() < 0) {
         if (id && *id)
             XtRemoveInput(*id);
         OVwDone();
         return;

     /*
     **  If the file descriptor has gone bad, remove it from
     ** the Xt select loop.
     */
     if (fd && *fd >= 0) {
         if (fstat(*fd, &buf) < 0) {
             if (id && *id)
                 XtRemoveInput(*id);
             OVwDone();
             return;
         }
     }
}

void
OVwXtMainLoop()
{
     int fd = OVwFileDescriptor();
     if (fd >= 0)
         XtAddInput (fd, (XtPointer)XtInputReadMask, __OVwInputCB, NULL);
     while (1)  {
         if (OVwPending())
             OVwProcessEvent();
         else
             XtProcessEvent(XtIMAll);
     }
}
```

Implementation Specifics

OVwXtMainLoop and OVwXtAppMainLoop support single-byte and multi-byte character code sets.

Libraries

When compiling a program that uses OVwXtMainLoop or OVwXtAppMainLoop, you need to link to the following libraries:

- /usr/OV/lib/libovw.a
- /usr/OV/lib/libov.a
- /usr/OV/lib/libntl.a

Related Information

- See ovw(1).
- See “OVwError(3)” on page 688.
- See “OVwInit(3)” on page 741.
- See “OVwApiIntro(5)” on page 560.

SnmpCleanup(3)

Purpose

Deallocates all WinSNMP application resources

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpCleanup (void);
```

Description

The SnmpCleanup function informs the implementation that the calling application is disconnecting and no longer requires the open resources allocated to it by the implementation. The implementation deallocates all resources allocated to the application, unless they have also been allocated to other active applications.

Note: It is the responsibility of an application to use the respective SnmpFree<*xxx*> functions to free specific resources created on its behalf and to use SnmpClose to clean-up after every session opened with SnmpCreateSession.

If an application must perform an emergency exit and call SnmpCleanup without performing the SnmpFree and SnmpClose steps, an implementation must cleanup all resources under its control which were created on behalf of or otherwise allocated to that application. Even in this emergency situation, however, the application *must* call SnmpCleanup to enable this functionality in the implementation.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS. Every subsequent WinSNMP API function call, until another successful SnmpStartup call, returns SNMPAPI_FAILURE with SnmpGetLastError or SnmpGetLastErrorStr set to report SNMPAPI_NOT_INITIALIZED.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. The application should behave as though it had returned SNMPAPI_SUCCESS. As an additional step the application could call SnmpGetLastError or SnmpGetLastErrorStr to ascertain the reason for failure.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpClose(3)

Purpose

Closes a WinSNMP session

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS  SnmpClose (
   IN HSNMP_SESSION session);
```

Description

The SnmpClose function causes the implementation to deallocate or close memory, resources, communications mechanisms, and data structures associated with the specified session on behalf of the calling application.

Closing a session on asynchronous requests that are outstanding causes those requests to be discarded by the implementation.

Note: A well-behaved WinSNMP application calls SnmpClose for each session opened by SnmpCreateSession. When an emergency exit is required of the application, it *must* at least call SnmpCleanup. A well-behaved WinSNMP implementation *must* react to an SnmpCleanup call as though it were a series of SnmpClose calls for each open session allocated to the calling application.

Parameters

session Identifies the handle of the session to close.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use SnmpGetLastError or SnmpGetLastErrorStr to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpContextToStr(3)

Purpose

Retrieves a textual context descriptor corresponding to the given WinSNMP context

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpContextToStr (
   IN HSNMP_CONTEXT   context,
   OUT smiLPOCTETS    string);
```

Description

The SnmpContextToStr function populates an smiOCTETS descriptor with a context value appropriate to the entity/context translation mode in effect at the time of execution.

The application provides only the address of a valid smiOCTETS descriptor structure as the string parameter. The implementation, upon successful execution of the SnmpContextToStr function, populates the len and ptr members of the descriptor. The application must call the SnmpFreeDescriptor, when appropriate, to enable the implementation to free the memory resources so consumed.

Note: Strings referenced in descriptors (such as an smiOCTETS structure) do not require a NULL terminating byte. Applications should not expect a NULL-terminated string to be returned in an OUTput smiOCTETS parameter.

When the translation mode in effect is SNMPAPI_TRANSLATED, the implementation returns the user-friendly textual name of this context value from the local database. When a user-friendly name does not exist, the function returns either SNMPAPI_UNTRANSLATED_V1 or SNMPAPI_UNTRANSLATED_V2 (see the following descriptions), depending upon whether the context value is known to be an SNMPv1 or SNMPv2 construct.

NetView for AIX Implementation Note

The WinSNMP local database is currently implemented as the configuration file: /usr/OV/conf/snmpv2.conf. Users may define SNMPv2C and secure SNMPv2USEC agents by inserting entries into this file.

When the translation mode in effect is SNMPAPI_UNTRANSLATED_V1 and the subject context value is an SNMPv1 construct, the implementation returns the raw community string (which may contain non-printable byte values). When the subject context value is an SNMPv2 construct, the implementation behaves as though the entity/context translation mode setting were SNMPAPI_UNTRANSLATED_V2 for the purposes of this call only.

When the translation mode in effect is SNMPAPI_UNTRANSLATED_V2 and the subject context value is an SNMPv2 construct, the implementations returns the raw Context ID (in textual form). When the subject entity is an SNMPv1 construct, the implementation behaves as though the entity/context translation mode setting were SNMPAPI_UNTRANSLATED_V1 for the purposes of this call only.

NetView for AIX Implementation Note

The SNMP_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C” model.

When the translation mode in effect is SNMPAPI_UNTRANSLATED_V2, the IBM implementation does *not* return a Context ID (obsolete) as described above but instead returns the raw community string as it does for the SNMPAPI_UNTRANSLATED_V1 mode.

Parameters

context Identifies the handle specifying a context.

string Identifies a pointer to an smiOCTETS descriptor buffer that receives the string which identifies the context.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_CONTEXT_INVALID	Indicates that the context handle is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpEntityToStr(3)” on page 884.
- See “SnmpFreeContext(3)” on page 886.
- See “SnmpFreeEntity(3)” on page 890.
- See “SnmpStrToContext(3)” on page 945.
- See “SnmpStrToEntity(3)” on page 948.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context translation modes and the section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpCountVbl(3)

Purpose

Counts the number of varbinds in a varbindlist structure

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS       SnmpCountVbl (
   IN HSNMP_VBL      vbl);
```

Description

The SnmpCountVbl function counts the number of varbinds in the varbindlist identified by the vbl input parameter.

The value returned when the result is SNMPAPI_SUCCESS represents the maximum “index” value in the SnmpGetVb and SnmpSetVb functions.

Parameters

vbl Identifies the subject varbindlist.

Return Values

When the function is successful, the return value is the count of varbinds in the varbindlist.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.
SNMPAPI_NOOP	Indicates that the vbl resources contained no varbinds at this time.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpGetVb(3)” on page 911.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpCreatePdu(3)

Purpose

Creates an SNMP protocol data unit (PDU) for use in subsequent communication requests

Syntax

```
#include  <WinSNMP.h>

HSNMP_PDU        SnmpCreatePdu (
   IN HSNMP_SESSION    session,
   IN smiINT           PDU_type,
   IN smiINT32         request_id,
   IN smiINT           error_status,  -- "non_repeaters" for BulkPDU
   IN smiINT           error_index,   -- "max_repetitions" for BulkPDU
   IN HSNMP_VBL        vbl);
```

Description

The SnmpCreatePdu function allocates and initializes an SNMP protocol data unit for subsequent use in SnmpSendMsg, SnmpEncodeMsg, and other functions.

All input parameters to SnmpCreatePdu must be present. If all input parameters (other than the session parameter) are Null, the created PDU defaults to the following attributes:

PDU_type:	SNMP_PDU_GETNEXT
request_id:	<WinSNMP-generated value>
error_status:	SNMP_ERROR_NOERROR
error_index:	0
vbl:	NULL

After completing operations with the created PDU, the SnmpFreePdu function should be called to release the resources allocated to the PDU by the SnmpCreatePdu function.

Parameters

session Identifies the handle of the allocating session.

PDU_type

NULL or one of the following values. If NULL, the WinSNMP implementation supplies SNMP_PDU_GETNEXT.

SNMP_PDU_GET	Indicates a GetRequest PDU
SNMP_PDU_GETNEXT	Indicates a GetNextRequest PDU
SNMP_PDU_GETBULK	Indicates an SNMPv2 GetBulkRequest PDU
SNMP_PDU_V1TRAP	Indicates an SNMPv1 Trap PDU
SNMP_PDU_SET	Indicates a SetRequest PDU
SNMP_PDU_INFORM	Indicates an SNMPv2 InformRequest PDU
SNMP_PDU_RESPONSE	Indicates a Response PDU
SNMP_PDU_TRAP	Indicates an SNMPv2 Trap PDU

request_id

An application-supplied value used to identify the PDU or NULL, in which case the WinSNMP implementation supplies a value.

error_status
Ignored (and may be NULL) on input for all PDU types except SNMP_PDU_GETBULK, in which case it represents the value for non_repeaters. For all other PDU types, the WinSNMP implementation supplies SNMP_ERROR_NOERROR.

error_index
Ignored (and may be NULL) on input for all PDU types except SNMP_PDU_GETBULK, in which case it represents the value for max_repetitions. The WinSNMP implementation returns 0 (zero) for all other PDU types.

vbl A handle to a varbindlist data structure (or NULL).

Return Values

When the function is successful, the return value identifies the created SNMP PDU handle.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session handle is not valid.
SNMPAPI_PDU_INVALID	Indicates that the PDU_type value is not valid.
SNMPAPI_VBL_INVALID	Indicates that the vbl is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDuplicatePdu(3)” on page 878.
- See “SnmpGetPduData(3)” on page 900.
- See “SnmpFreePdu(3)” on page 892.
- See “SnmpSetPduData(3)” on page 930.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP PDU functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpCreateSession(3)

Purpose

Creates a WinSNMP session and initializes resources for subsequent communication functions

Syntax

```
#include  <WinSNMP.h>

HSNMP_SESSION SnmpCreateSession (IN HWND hWnd,
                                 IN UINT wMsg,
                                 IN CALLBACK fCallBack,
                                 IN LPVOID lpClientData);
```

Description

The SnmpCreateSession function enables the implementation to allocate and initialize memory, resources, communications mechanisms, and data structures for the application. The application continues to use the “session identifier” returned by the implementation in subsequent WinSNMP function calls to facilitate resource accounting on a per session basis. This mechanism enables the implementation to perform an orderly release of resources in response to a subsequent SnmpClose function call for a given session.

An application can open multiple sessions. A successful SnmpCreateSession call always returns a unique session handle (with respect to all other currently open sessions for the calling application).

Two popular modes of window notification are possible using SnmpCreateSession, notification messages and notification callbacks. The hWnd parameter specifies the window handle to be notified when an asynchronous request completes or trap/notification occurs. If the implementation supports notification messages, the wMsg parameter specifies the message number that the window is sent. If the implementation supports notification callbacks, the fCallBack parameter is a pointer to the applications callback function which will be executed to inform the window that an asynchronous request has completed or failed. Although an implementation may support both notification modes, the application must indicate which mode the new WinSNMP session is to use. The wMsg and fCallBack parameters are therefore mutually exclusive. If one is specified, the other must be set to NULL.

NetView for AIX Implementation Note

The IBM implementation supports notification callbacks *only*, since this is the method used by the AIX X-Window system. This implementation interprets the hWnd parameter as an X-Window application context which identifies the window to be notified. A NULL hWnd parameter indicates that the calling application is not X-Window based (such as an AIX command line application). The HWND data type is equivalent to the XtAppContext type as returned by numerous X-Window Intrinsics functions (for example, XtAppInitialize(), XtCreateApplicationContext(), XtDisplayToApplicationContext(), and XtWidgetToApplicationContext()).

Since the notification message model is not supported, the wMsg parameter is not used by the IBM implementation. The user should set this parameter to NULL.

The format of the user's callback function should be as follows:

```
CALLBACK <fCallBack> (IN HSNMP_SESSION session,
                      IN HWND hWnd,
                      IN UINT wMsg,
                      IN WPARAM wParam,
                      IN LPARAM lParam,
                      IN LPVOID lpClientData);
```

The session parameter contains the owning session ID of the request being processed. The hWnd parameter contains the X-Window ID (XtAppContext) which corresponds to the owning session. Since the notification message model is not supported, the the wMsg parameter is zero (0). If this is a failed request, the wParam parameter contains one of the error codes listed for “SnmpRecvMsg(3)” on page 919. The lParam parameter contains the request ID of the request being processed. The lpClientData parameter points to generic data supplied by the user.

If the callback is executed with wParam set to NULL (indicating a successful request), the function should proceed to SnmpRecvMsg to retrieve the subject PDU for immediate or subsequent processing. If the callback is executed with wParam set to a non-NULL error code, the function may want to notify the user or perform some other corrective action.

Note: A well-behaved WinSNMP application will call SnmpClose for each session opened by SnmpCreateSession. When an emergency exit is required of the application, it *must* at least call SnmpCleanup. The implementation reacts as though a series of SnmpClose calls for each open session allocated to the calling application has been received.

Parameters

hWnd Identifies the application's notification window (X-Window application context).

wMsg This parameter is ignored by NetView for AIX. Should be set to NULL.

fCallBack Identifies the application's callback function.

lpClientData
Pointer to generic application data.

Return Values

When the function is successful, the return value is a handle which identifies the WinSNMP session opened by the implementation on behalf of the calling application.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_HWND_INVALID	Indicates that the hWnd parameter is not a valid window handle.
SNMPAPI_MSG_INVALID	Indicates that the wMsg parameter is not a valid message value.
SNMPAPI_MODE_INVALID	Indicates that the combination of values passed does not identify a valid notification mode.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about sessions and the section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpCreateVbl(3)

Purpose

Creates and initializes a new varbindlist structure

Syntax

```
#include  <WinSNMP.h>

HSNMP_VBL             SnmpCreateVbl (
   IN HSNMP_SESSION   session,
   IN smiLPCOID       name,
   IN smiLPCVALUE     value);
```

Description

The SnmpCreateVbl function creates a new varbindlist structure for the calling application. If the name and value parameters are not NULL, SnmpCreateVbl uses them to construct the initial varbind member of the varbindlist.

If the name parameter is not NULL and the value parameter is NULL, the varbindlist is initialized, with the value set to NULL and with the syntax of SNMP_SYNTAX_NULL. If the name parameter is NULL, the varbindlist is not initialized, and the value parameter is ignored.

Each call to SnmpCreateVbl must be matched with a corresponding call to SnmpFreeVbl to release the resources associated with the varbindlist. A memory leak results if a variable used to hold an HSNMP_VBL value returned by SnmpCreateVbl (or SnmpDuplicateVbl) is reused for a subsequent SnmpCreateVbl (or SnmpDuplicateVbl) operation before it has been passed to SnmpFreeVbl.

Parameters

session Identifies the handle of the allocating session.

name If not NULL, points to an OID for initialization of the varbindlist.

value If not NULL, points to a value for initialization of the varbidlist.

Return Values

When the function is successful, the return value identifies a handle to the newly created varbindlist structure.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_SESSION_INVALID	Indicates that the session handle is not valid.
SNMPAPI_OID_INVALID	Indicates that the name parameter referenced an OID structure that is not valid.
SNMPAPI_SYNTAX_INVALID	Indicates that the syntax field of the value parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpGetVb(3)” on page 911.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpDecodeMsg(3)

Purpose

Decodes the specified SNMP message

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpDecodeMsg (
   IN HSNMP_SESSION          session,
   OUT LPHSNMP_ENTITY        srcEntity,
   OUT LPHSNMP_ENTITY        dstEntity,
   OUT LPHSNMP_CONTEXT       context,
   OUT LPHSNMP_PDU           pdu,
   IN smiLPCOCTETS           msgBufDesc);
```

Description

The SnmpDecodeMsg function is the converse of the SnmpEncodeMsg function. It takes as input a session identifier and a far pointer to an smiOCTETS structure which describes an encoded or serialized SNMP message to be decoded into its constituent components. The session identifier is required because new resources are created by the implementation and allocated to the application as a result of calling this function, if it is successful. The msgBufDesc input parameter consists of two elements: the len member identifies the maximum number of bytes to process and the ptr member points to the encoded or serialized SNMP message to decode.

The SnmpDecodeMsg function is meant to be symmetrical with the SnmpEncodeMsg function. Refer to “SnmpEncodeMsg(3)” on page 882 for additional insight into the operation and possible failure modes of the SnmpDecodeMsg function.

Parameters

session Identifies the session that performs the operation.

srcEntity Identifies the subject management entity.

dstEntity Identifies the target management entity.

context Identifies the target context of interest.

PDU Identifies the SNMP protocol data unit.

msgBufDesc
Identifies the buffer holding the encoded SNMP message.

Return Values

When the function is successful, the return value is the actual number of bytes decoded. This may be equal to or less than the len member of the msgBufDesc input parameter. Also, upon success, the SnmpDecodeMsg returns handle values in the srcEntity, dstEntity, context, and pdu output parameters. These resources are to be freed by the application using the appropriate SnmpFree<*xxx*> functions, or by the implementation in response to an SnmpClose or SnmpCleanup function call.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and SnmpGetLastError or SnmpGetLastErrorStr is set to report one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that a session parameter is not valid.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.
SNMPAPI_CONTEXT_INVALID	Indicates that the context parameter is not valid.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.
SNMPAPI_OUTPUT_TRUNCATED	Indicates that the buffer was too small. That is, len bytes of ptr were consumed before reaching the end of the encoded message; no output parameters are created.
SNMPAPI_MESSAGE_INVALID	Indicates that the SNMP message described by the msgBufDesc parameter is not valid; no output resources are created.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpDeleteVb(3)

Purpose

Deallocates resources associated with the specified WinSNMP varbindlist

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpDeleteVb (
   IN HSNMP_VBL   vbl,
   IN smiUINT32   index);
```

Description

The SnmpDeleteVb function removes a varbind entry from a varbindlist.

Valid values for the index parameter come from the SnmpCountVbl function and from the error_index component of GetResponse PDUs returned with the SnmpRecvMsg function. These values range from 1 to *n*, where *n* is the total number of varbinds in the varbindlist.

A typical use for this function is when a GetResponse PDU includes an SNMP error and the user elects to resubmit the original request PDU without the offending varbind.

Following a successful SnmpDeleteVb operation, any varbinds that previously came after the deleted varbind, logically “move up” in the varbindlist, this means that their index values decrease by one position and the total number of varbinds in the varbindlist, as returned by SnmpCountVbl, also decrease by one.

It is legal to end up with an empty varbindlist by executing SnmpDeleteVb (hVBL, 1) on the last remaining varbind in a varbindlist. In this case, the varbindlist itself (as a HANDLE'd object) is valid and must eventually be released through SnmpFreeVbl.

Sample pseudo-code for SnmpDeleteVb:

```
-- Omitting error-checking the function calls for clarity's sake...
nReqID = SnmpRecvMsg (session, &rSrc, &rDst, &rCtx, &rPDU);
SnmpGetPduData (rPDU, &rType, &rReqid, &rErrstat, &rErridx, &rVBL);
-- Assuming type == GetResponse-PDU and
-- Assuming error_status != SNMP_ERROR_NOERROR...
-- Assuming the error is something we cannot or do not want to fix...
    -- If error_index == 1, do an SnmpCountVbl (3);
    -- if count <= 1 follow another strategy (like SnmpFreeVbl (sVBL))
-- Assuming error_index > 1 || count > 1...
SnmpDeleteVb (sVBL, rErridx);
-- And assuming we want to re-try the SNMP operation
-- ...with a new Request_ID just for good measure...
sReqid++;
SnmpSetPduData (sPDU, NULL, &sReqid, NULL, NULL, &sVBL);
SnmpSendMsg (session, sSrc, sDst, sCtx, sPDU);
-- No need for the received PDU or VBL any longer...
SnmpFreePdu (rPDU);
SnmpFreeVbl (rVBL);
-- Go back to doing what we were doing before all of this started...
```

Parameters

vbl Identifies the target varbindlist.

index Identifies the position of the subject varbind within the varbindlist.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.
SNMPAPI_INDEX_INVALID	Indicates that the index parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpGetVb(3)” on page 911.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpDuplicatePdu(3)

Purpose

Duplicates the specified PDU

Syntax

```
#include  <WinSNMP.h>

HSNMP_PDU               SnmpDuplicatePdu (
   IN HSNMP_SESSION     session,
   IN HSNMP_PDU         PDU);
```

Description

The SnmpDuplicatePdu function duplicates an SNMP PDU structure identified by the PDU parameter.

After using the duplicated message, the SnmpFreePdu function should be called to release the resources allocated to the PDU by the SnmpDuplicatePdu function.

Note: The handle returned by a successful call to SnmpDuplicatePdu is unique among active PDU handles, at least within the calling application.

Parameters

session Identifies the handle of the allocating session.

PDU Identifies the SNMP PDU to duplicate.

Return Values

When the function is successful, the return value is a handle which identifies the new (duplicated) SNMP PDU.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session handle is not valid.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCreatePdu(3)” on page 867.
- See “SnmpFreePdu(3)” on page 892.
- See “SnmpGetPduData(3)” on page 900.
- See “SnmpSetPduData(3)” on page 930.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP PDU functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpDuplicateVbl(3)

Purpose

Duplicates the specified varbindlist structure

Syntax

```
#include  <WinSNMP.h>

HSNMP_VBL              SnmpDuplicateVbl (
   IN HSNMP_SESSION    session,
   IN HSNMP_VBL        vbl);
```

Description

The SnmpDuplicateVbl function creates a new varbindlist structure for the specified session in the calling application and initializes it with a copy of the input vbl parameter (which may be empty).

Every call to SnmpDuplicateVbl must be matched with a corresponding call to SnmpFreeVbl to release the resources associated with the varbindlist. A memory leak results if a variable used to hold an HSNMP_VBL value returned by SnmpDuplicateVbl (or SnmpCreateVbl) is reused for a subsequent SnmpDuplicateVbl (or SnmpCreateVbl) operation before it has been passed to SnmpFreeVbl.

Note: The handle returned by a successful call to SnmpDuplicateVbl is unique among active VBL handles, at least within the calling application.

Parameters

session Identifies the handle of the allocating session.

vbl Identifies the varbindlist to be duplicated.

Return Values

When the function is successful, the return value identifies a newly created varbindlist structure.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session handle is not valid.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpGetVb(3)” on page 911.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpEncodeMsg(3)

Purpose

Encodes an SNMP message without sending it

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpEncodeMsg (
   IN HSNMP_SESSION    session,
   IN HSNMP_ENTITY     srcEntity,
   IN HSNMP_ENTITY     dstEntity,
   IN HSNMP_CONTEXT    context,
   IN HSNMP_PDU        pdu,
   OUT smiLPOCTETS     msgBufDesc);
```

Description

The SnmpEncodeMsg routine takes as its first five input parameters the same parameters passed to SnmpSendMsg. The implementation uses these parameters to form an SNMP “message” as though they had arrived through the SnmpSendMsg function. The implementation does not, however, attempt to transmit the resulting message to the dstEntity parameter. Instead, it uses the msgBufDesc parameter to return to the application the encoded or serialized SNMP message that it would have transmitted to the dstEntity parameter if SnmpSendMsg had been called.

The member elements of the smiOCTETS structure pointed to by the msgBufDesc structure are ignored and overwritten by the implementation upon a successful execution of this function.

The application must eventually call the SnmpFreeDescriptor function to enable the implementation to free any resources that might have been allocated to populate the ptr member of the msgBufDesc structure.

When the normal integrity checks performed for SnmpSendMsg are unsuccessful for any of the first five input parameters, the return value is SNMPAPI_FAILURE and SnmpGetLastError or SnmpGetLastErrorStr is set to return the appropriate extended error code.

Parameters

session Identifies the session that performs the operation.

srcEntity Identifies the subject management entity.

dstEntity Identifies the target management entity.

context Identifies the target context of interest.

PDU Identifies the SNMP PDU containing the requested operation.

msgBufDesc
Identifies the variable to receive the encoded SNMP message.

Return Values

When the function is successful, the return value is the length, in bytes, of the encoded SNMP message. (This value is also in the len member of the msgBufDesc output parameter.)

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use SnmpGetLastError or SnmpGetLastErrorStr to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that a session parameter is not valid.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.
SNMPAPI_CONTEXT_INVALID	Indicates that the context parameter is not valid.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpEntityToStr(3)

Purpose

Returns a textual string for the given WinSNMP entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpEntityToStr (
   IN HSNMP_ENTITY     entity,
   IN smiUINT32        size,
   OUT LPSTR           string);
```

Description

The SnmpEntityToStr function returns a string value identifying an entity.

When the translation mode in effect is SNMPAPI_TRANSLATED, the implementation returns the user-friendly textual name of this entity from the local database. When a user-friendly name does not exist, the function returns either SNMPAPI_UNTRANSLATED_V1 or SNMPAPI_UNTRANSLATED_V2 (see the following descriptions), depending upon whether the entity is known to be SNMPv1 or SNMPv2.

NetView for AIX Implementation Note

The WinSNMP local database is currently implemented as the configuration file: /usr/OV/conf/snmpv2.conf. Users may define SNMPv2C and secure SNMPv2USEC agents by inserting entries into this file.

When the setting is SNMPAPI_UNTRANSLATED_V1 and the subject entity is an SNMPv1 creature, the implementation returns the transport address of the entity (in textual form). When the subject entity is an SNMPv2 creature, the implementation behaves as though the entity/context translation mode setting were SNMPAPI_UNTRANSLATED_V2 for the purposes of this call only.

When the setting is SNMPAPI_UNTRANSLATED_V2 and the subject entity is an SNMPv2 creature, the implementation returns the PartyID of the entity (in textual form). When the subject entity is an SNMPv1 creature, the implementation behaves as though the entity/context translation mode setting were SNMPAPI_UNTRANSLATED_V1 for the purposes of this call only.

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C“ model.

When the translation mode in effect is SNMPAPI_UNTRANSLATED_V2, the IBM implementation does *not* return a PartyID (obsolete) as described above, but instead returns the textual form of the transport address of the entity as it does for the SNMPAPI_UNTRANSLATED_V1 mode.

Parameters

entity Identifies the handle specifying an entity.

size The size of the buffer that the application is providing to contain the string.

string Points to a buffer that receives the NULL-terminated string that identifies the management entity.

Return Values

When the function is successful, the return value is the number of bytes, including the NULL terminating byte, in the output string. This return value may be less than or equal to the size parameter, but not greater.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.

SNMPAPI_ALLOC_ERROR Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_ENTITY_INVALID Indicates that the entity parameter is unknown.

SNMPAPI_OUTPUT_TRUNCATED
Indicates that the buffer was too small.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpContextToStr(3)” on page 862.
- See “SnmpFreeContext(3)” on page 886.
- See “SnmpFreeEntity(3)” on page 890.
- See “SnmpStrToContext(3)” on page 945.
- See “SnmpStrToEntity(3)” on page 948.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context translation modes and the section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpFreeContext(3)

Purpose

Deallocates resources for the specified WinSNMP context

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpFreeContext (
   IN HSNMP_CONTEXT    context);
```

Description

The SnmpFreeContext function releases resources associated with a context returned by the SnmpStrToContext function.

Un-freed resources created on behalf of the application are freed by the implementation upon execution of an associated SnmpClose function or upon execution of an SnmpCleanup function. Nonetheless, a well-behaved WinSNMP application individually frees all such resources using the atomic "free" functions. This eliminates or, at least, minimizes batch-like loads on the implementation, so that other applications can be serviced in a timely fashion.

Parameters

context Identifies a context handle to be released.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_CONTEXT_INVALID	Indicates that the context parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpContextToStr(3)” on page 862.
- See “SnmpEntityToStr(3)” on page 884.
- See “SnmpFreeEntity(3)” on page 890.
- See “SnmpStrToContext(3)” on page 945.
- See “SnmpStrToEntity(3)” on page 948.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpFreeDescriptor(3)

Purpose

Deallocates resources associated with the specified WinSNMP descriptor object

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpFreeDescriptor (
   IN smiUINT32         syntax,
   IN smiLPOPAQUE       descriptor);
```

Description

The SnmpFreeDescriptor function is used by the application to inform the implementation that it no longer requires access to a WinSNMP “descriptor object” that had been “populated” earlier on its behalf by the implementation.

WinSNMP descriptor objects are either smiOID or smiOCTETS structures (or equivalents, such as smiIPADDR and smiOPAQUE) and consist of a len member and a ptr member.

These objects are populated by the implementation on behalf of the application in response to any OUT parameter of type smiOID, smiOCTETS, and smiVALUE. (Any smiVALUE structure may or may not contain an smiOID or smiOCTETS structure in its value member upon return from SnmpGetVb, as indicated by the associated syntax member of the smiVALUE structure.) In addition to SnmpGetVb, the following functions also result in the implementation populating a descriptor object for the application: SnmpContextToStr, SnmpStrToOid, SnmpOidCopy, and SnmpEncodeMsg. Others may be added later.

Applications should not attempt to free memory returned in the ptr member of descriptor objects that have been populated by the implementation. The method of memory allocation and, consequently, deallocation, for these variables is private to the implementation, and hidden from the application except for the SnmpFreeDescriptor interface described in this section.

The syntax parameter can be used by implementations to distinguish among different varieties of descriptor objects, if necessary. The SNMPAPI_OPERATION_INVALID error can be returned if, for example, the descriptor parameter does not satisfy implementation-specific requirements. For example, the implementation can recognize that the ptr member does not identify an allocation that it has made on behalf of the calling application or if the indicated allocation had already been released by the application in a prior call to the SnmpFreeDescriptor.

Parameters

syntax Identifies the “syntax” (data type) of the target descriptor.

descriptor Identifies the target descriptor object.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and SnmpGetLastError or SnmpGetLastErrorStr is set to report one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SYNTAX_INVALID	Indicates that the syntax parameter is not valid.
SNMPAPI_OPERATION_INVALID	Indicates that the descriptor parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP descriptors and the section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpFreeEntity(3)

Purpose

Deallocates resources for the specified WinSNMP entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpFreeEntity (
   IN HSNMP_ENTITY      entity);
```

Description

The SnmpFreeEntity function releases resources associated with an entity returned by the SnmpStrToEntity function.

Un-freed resources created on behalf of the application are freed by the implementation upon execution of an associated SnmpClose function or upon execution of an SnmpCleanup function. Nonetheless, a well-behaved WinSNMP application individually frees all such resources using the atomic "free" functions. This eliminates or, at least, minimizes batch-like loads on the implementation, so that other applications can be serviced in a timely fashion.

Parameters

entity Identifies the entity handle to be released.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_ENTITY_INVALID	Indicates that the entity parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpContextToStr(3)” on page 862.
- See “SnmpEntityToStr(3)” on page 884.
- See “SnmpFreeContext(3)” on page 886.
- See “SnmpStrToContext(3)” on page 945.
- See “SnmpStrToEntity(3)” on page 948.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpFreePdu(3)

Purpose

Deallocates resources for the specified WinSNMP PDU

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpFreePdu (
   IN HSNMP_PDU   PDU);
```

Description

The SnmpFreePdu function releases resources associated with a PDU previously created by the SnmpCreatePdu or SnmpDuplicatePdu function.

Un-freed resources created on behalf of the application are freed by the implementation upon execution of an associated SnmpClose function or upon execution of an SnmpCleanup function. Nonetheless, a well-behaved WinSNMP application individually frees all such resources using the atomic "free" functions. This eliminates or, at least, minimizes batch-like loads on the implementation, so that other applications can be serviced in a timely fashion.

Varbinds and VarBindLists are reusable independently of any given PDU. In WinSNMP, a varbind does not exist outside of a varbindlist (even if the latter consists of only a single varbind). There is a separate atomic function, SnmpFreeVbl, that deallocates varbindlist resources. Upon execution of SnmpFreePdu, the WinSNMP implementation must free *internal* resources allocated to VBLs for that PDU. The internal resources are different from the HSNMP_VBL resources requested and held by a session in the calling application.

Parameters

PDU Identifies the SNMP protocol data unit to be freed.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCreatePdu(3)” on page 867.
- See “SnmpDuplicatePdu(3)” on page 878.
- See “SnmpGetPduData(3)” on page 900.
- See “SnmpSetPduData(3)” on page 930.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP PDU functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpFreeVbl(3)

Purpose

Deallocates resources associated with the specific VBL

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpFreeVbl (
   IN HSNMP_VBL   vbl);
```

Description

The SnmpFreeVbl function releases resources associated with a varbindlist structure previously allocated by SnmpCreateVbl or SnmpDuplicateVbl. It is the responsibility of the WinSNMP applications to free varbindlist resources allocated through calls to SnmpCreateVbl and SnmpDuplicateVbl.

Every call to SnmpCreateVbl must be matched with a corresponding call to SnmpFreeVbl to release the resources associated with the varbindlist. A memory leak results if a variable used to hold an HSNMP_VBL value returned by SnmpCreateVbl is reused for a subsequent SnmpCreateVbl operation before it has been passed to SnmpFreeVbl.

Un-freed resources created on behalf of the application are freed by the implementation upon execution of an associated SnmpClose function or upon execution of an SnmpCleanup function. Nonetheless, a well-behaved WinSNMP application individually frees all such resources using the atomic "free" functions. This eliminates or, at least, minimizes batch-like loads on the implementation, so that other applications can be serviced in a timely fashion.

Parameters

vbl Identifies the varbindlist to be released.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpGetVb(3)” on page 911.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetLastError(3)

Purpose

Indicates why the last WinSNMP operation failed

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS          SnmpGetLastError (
   IN HSNMP_SESSION     session);
```

Description

The SnmpGetLastError function returns an indication of why the last WinSNMP operation executed by the application was unsuccessful.

This function should be called immediately after each unsuccessful API call, as the error information will be overwritten by the next unsuccessful API call.

The session input parameter is provided to facilitate accommodation of multi-threaded Windows operating environments. Single-threaded applications can always pass a NULL session value and retrieve the last error information for the overall application.

SnmpGetLastError must be able to return a value to a WinSNMP application when: SnmpStartup is unsuccessful, before any sessions are created with SnmpCreateSession, after all sessions are closed with SnmpClose, or the application disconnects from the implementation with the SnmpCleanup function.

Parameters

session Indicates the session for which error information is requested. If NULL, the application-wide error information is returned.

Return Values

This function returns the last WinSNMP error that occurred for the indicated session or for the application (task) if the session is NULL (for example, when SnmpStartup is unsuccessful).

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.

- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about error handling and the section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetLastErrorStr(3)

Purpose

Provides a textual description of why the last WinSNMP operation failed

Syntax

```
#include  <WinSNMP.h>
#include  <IBMwsnmp.h>

smiLPBYTE  SnmpGetLastErrorStr(
       IN HSNMP_SESSION  session);
```

Description

The SnmpGetLastErrorStr function is an IBM extension to the NetView for AIX WinSNMP API. It is similar to the SnmpGetLastError function, but returns a textual description of the last WinSNMP failure (instead of an error code).

This function (or SnmpGetLastError) should be called immediately after one of the other WinSNMP functions fail, as the error information may be overwritten by a subsequent failure.

The session input parameter is provided to facilitate accommodation of multi-threaded Windows operating environments. Single-threaded applications can always pass a NULL session value and retrieve the last error information for the overall application.

SnmpGetLastErrorStr must be able to return a value to a WinSNMP application when: SnmpStartup is unsuccessful, before any sessions are created with SnmpCreateSession, after all sessions are closed with SnmpClose, or the application disconnects from the implementation with the SnmpCleanup function.

Parameters

session Indicates the session for which error information is requested. If NULL, the application-wide error information is returned.

Return Values

This function returns a pointer to a data buffer which contains a textual description of the last WinSNMP failure for the specific session or for the entire application if the session is NULL.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

For further information see the following:

- The section about error handling and the section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetPduData(3)

Purpose

Extracts data from the specified PDU

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS       SnmpGetPduData (
   IN HSNMP_PDU      PDU
   OUT smiLPINT      PDU_type,
   OUT smiLPINT32    request_id,
   OUT smiLPINT      error_status,  -- "non_repeaters" for GetBulkRequest-PDU
   OUT smiLPINT      error_index,   -- "max_repetitions" for GetBulkRequest-PDU
   OUT LPHSNMP_VBL   vbl);
```

Description

The SnmpGetPduData function extracts selected data elements from the specified PDU and copies them to the respective locations given as corresponding output parameters.

All output parameters must be supplied to the function call, and any (or all) of them may be NULL. No values are returned for output parameters passed as NULL.

On a successful return and if the parameter was not NULL, the PDU_type contains one of the following values:

SNMP_PDU_GET	Indicates a GetRequest PDU
SNMP_PDU_GETNEXT	Indicates a GetNextRequest PDU
SNMP_PDU_GETBULK	Indicates an SNMPv2 GetBulkRequest PDU
SNMP_PDU_V1TRAP	Indicates an SNMPv1 Trap PDU
SNMP_PDU_SET	Indicates a SetRequest PDU
SNMP_PDU_INFORM	Indicates an SNMPv2 InformRequest PDU
SNMP_PDU_RESPONSE	Indicates a Response PDU
SNMP_PDU_TRAP	Indicates an SNMPv2 Trap PDU

On a successful return and if the parameter was not NULL, the error_status contains one of the following values:

SNMP_ERROR_NOERROR	Specifies the noError error
SNMP_ERROR_TOOBIG	Specifies the tooBig error
SNMP_ERROR_NOSUCHNAME	Specifies the noSuchName error
SNMP_ERROR_BADVALUE	Specifies the badValue error
SNMP_ERROR_READONLY	Specifies the readOnly error
SNMP_ERROR_GENERR	Specifies the genErr error
SNMP_ERROR_NOACCESS	Specifies the noAccess error
SNMP_ERROR_WRONGTYPE	Specifies the wrongType error
SNMP_ERROR_WRONGLENGTH	Specifies the wrongLength error
SNMP_ERROR_WRONGENCODING	Specifies the wrongEncoding error

SNMP_ERROR_WRONGVALUE	Specifies the wrongValue error
SNMP_ERROR_NOCREATION	Specifies the noCreation error
SNMP_ERROR_INCONSISTENTVALUE	Specifies the inconsistentValue error
SNMP_ERROR_RESOURCEUNAVAILABLE	Specifies the resourceUnavailable error
SNMP_ERROR_COMMITFAILED	Specifies the commitFailed error
SNMP_ERROR_UNDOFAILED	Specifies the undoFailed error
SNMP_ERROR_AUTHORIZATIONERROR	Specifies the authorizationError error
SNMP_ERROR_NOTWRITABLE	Specifies the notWritable error
SNMP_ERROR_INCONSISTENTNAME	Specifies the inconsistentName error

As always, a well-behaved application must handle the case when an unexpected value (for example, PDU_type or error_status) might be returned by a procedure call.

It is very important to note that the HSNMP_VBL resource returned when the call is successful and the vbl parameter is not NULL represents a *new* varbindlist object. This is consistent with the primary purpose of the SnmpGetPduData function, which is to translate the elements of a *newly received* PDU and with the WinSNMP policy that says whenever the implementation returns a handled resource object to the application, it is a newly allocated object. For PDUs that an application has either created for sending or has already translated with SnmpGetPduData, the application is expected to "know" the values of the PDU components. Calling SnmpGetPduData with a non-NULL vbl parameter against an existing translated PDU is equivalent (with regards to the returned HSNMP_VBL object) to calling SnmpDuplicateVbl on the vbl object already attached to the PDU.

Calling SnmpGetPduData with a non-NULL vbl parameter against a *newly received* PDU has the effect of “attaching” the returned HSNMP_VBL object to the HSNMP_PDU object specified in the input pdu parameter. Calling SnmpGetPduData with a non-NULL vbl parameter against an existing (created or translated) PDU, results in a new HSNMP_VBL object as discussed in this section, but does not disturb the original HSNMP_VBL object value already attached to the PDU.

Parameters

PDU Identifies the SNMP protocol data unit.

PDU_type
If not NULL, points to an smiINT variable that receives the PDU_type of the PDU.

request_id
If not NULL, points to an smiINT32 variable that receives the request_id of the PDU.

error_status
If not NULL, points to an smiINT variable that receives the error_status (or non_repeaters) of the PDU.

error_index
If not NULL, points to an smiINT variable that receives the error_index (or max_repetitions) of the PDU.

vbl If not NULL, points to an HSNMP_VBL variable that receives the handle to the varbindlist of the PDU.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.
SNMPAPI_NOOP	Indicates that all output parameters were NULL.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCreatePdu(3)” on page 867.
- See “SnmpDuplicatePdu(3)” on page 878.
- See “SnmpFreePdu(3)” on page 892.
- See “SnmpSetPduData(3)” on page 930.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP PDU functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetRetransmitMode(3)

Purpose

Indicates the retransmission mode currently in effect

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpGetRetransmitMode (
   OUT smiLPUINT32 nRetransmitMode);
```

Description

The SnmpGetRetransmitMode function informs the calling application of the retransmission mode in effect at the time of the call.

Parameters

nRetransmitMode
: Identifies the pointer to variable to receive the current retransmission mode.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS. In this case, nRetransmitMode returns one of the following values:

SNMPAPI_ON
: Indicates that the implementation *is* doing retransmission.

SNMPAPI_OFF
: Indicates that the implementation is *not* doing retransmission.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. In this case, the value of nRetransmitMode is undefined and meaningless to the application, and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about polling and retransmission and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetRetry(3)

Purpose

Retrieves the retry value for the specified entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpGetRetry (
   IN HSNMP_ENTITY  hEntity,
   OUT smiLPUINT32  nPolicyRetry,
   OUT smiLPUINT32   nActualRetry);
```

Description

The SnmpGetRetry function returns current values for the retransmission retry value on a per-entity basis. The retry value is expressed as a unit count. The nPolicyRetry value refers to the retry value currently stored in the local database for the subject agent. The nActualRetry value refers to the last measured or estimated response retry count reported by the implementation.

Implementations may provide utilities to load initial retry count values for the retransmission policy on a per destination entity basis, or may automatically assign some initial default value. Subsequent modifications to this value are made by applications with the SnmpSetRetry function.

NetView for AIX Implementation Note

If not specified using SnmpSetRetry, the IBM implementation defaults an entity's retry value to 2.

Implementations may or may not return measured or estimated values for the Actual Retry parameter to the SnmpGetRetry function. In the latter case, the implementation should return zero (0).

Applications should monitor the Actual Retry value. If the value is near, equal to, or greater than the current Policy Retry value, the latter should be increased accordingly (or other corrective action taken).

NetView for AIX Implementation Note

The IBM implementation does *not* currently measure or estimate the Actual Retry value and therefore returns zero (0) for this parameter.

Parameters

hEntity Indicates the destination entity of interest.

nPolicyRetry
Points to a variable to receive the retry count value for this entity as stored in the implementation's local database.

nActualRetry
Points to a variable to receive the last measured or estimated response retry count from the destination agent.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about polling and retransmission and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetTimeout(3)

Purpose

Retrieves timeout information for the specified entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpGetTimeout (
   IN HSNMP_ENTITY     hEntity,
   OUT smiLPTIMETICKS  nPolicyTimeout,
   OUT smiLPTIMETICKS  nActualTimeout);
```

Description

The SnmpGetTimeout function returns current values for the retransmission timeout value on a per-entity basis. The timeout value is expressed in units of *hundredths of seconds*. The nPolicyTimeout value refers to the timeout value currently stored in the local database for the subject agent. The nActualTimeout value refers to the last measured or estimated response receipt interval reported by the implementation.

Implementations may provide utilities to load initial timeout values for the retransmission policy on a per destination entity basis, or may automatically assign some initial default value. Subsequent modifications to this value are made by applications with the SnmpSetTimeout function.

NetView for AIX Implementation Note

If not specified using SnmpSetTimeout, the IBM implementation defaults an entity's timeout to 5 seconds.

Implementations may or may not return measured or estimated values for the Actual Timeout parameter to the SnmpGetTimeout function. In the latter case, the implementation should return zero (0).

Applications should monitor the Actual Timeout value. If the value is near, equal to, or greater than the current Policy Timeout value, the latter should be increased accordingly (or other corrective action taken).

NetView for AIX Implementation Note

The IBM implementation does *not* currently measure or estimate the Actual Timeout value and therefore returns zero (0) for this parameter.

Parameters

hEntity Indicates the destination entity of interest.

nPolicyTimeout
Points to a variable to receive the timeout value for this entity as stored in the implementation's local database.

nActualTimeout
Points to a variable to receive the last measured or estimated response time interval from the destination agent.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about polling and retransmission and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetTranslateMode(3)

Purpose

Indicates the entity/context translation mode currently in effect

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpGetTranslateMode (
   OUT smiLPUINT32 nTranslateMode);
```

Description

The SnmpGetTranslateMode function informs the calling application as to the entity/context translation mode in effect at the time of the call.

Parameters

nTranslateMode
: Identifies the pointer to variable to receive the current translation mode.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS. In this case, nTranslateMode returns one of the following values:

SNMPAPI_TRANSLATED
: Indicates that the local database look-up translation mode will be used.

SNMPAPI_UNTRANSLATED_V1
: Indicates that the literal transport address and community string will be used.

SNMPAPI_UNTRANSLATED_V2
: Indicates that the literal SNMPv2 party and context IDs will be used.

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C“ model.

SNMPAPI_UNTRANSLATED_V2 no longer indicates that a literal party and context IDs (both obsolete) are used. Instead, the IBM implementation returns SNMPAPI_UNTRANSLATED_V2 to indicate that a literal transport address and community string are used.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. In this case, the value of nTranslateMode is undefined and meaningless to the application, and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context translation modes and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpGetVb(3)

Purpose

Extracts the varbind identified by the supplied index from a varbindlist structure

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpGetVb (
   IN HSNMP_VBL         vbl,
   IN smiUINT32         index,
   OUT smiLPOID         name,
   OUT smiLPVALUE       value);
```

Description

The SnmpGetVb function retrieves the object instance name and its associated value from the varbind identified by the index parameter. The SnmpGetVb function returns the object instance name in the descriptor pointed to by the name parameter and its associated value in the descriptor pointed to by the value parameter.

Valid values for the index parameter come from the SnmpCountVbl function and from the error_index component of GetResponse PDUs returned with the SnmpRecvMsg function. These values range from 1 to *n*, where *n* is the total number of varbinds in the varbindlist.

The member elements of the smiOID and smiVALUE structures pointed to by the name and value parameters are ignored on input and are overwritten by the implementation upon a successful execution of this function.

On a successful return, the syntax field of the value variable contains one of the following object syntax types:

SNMP_SYNTAX_INT32
SNMP_SYNTAX_OCTETS
SNMP_SYNTAX_OID
SNMP_SYNTAX_BITS
SNMP_SYNTAX_IPADDR
SNMP_SYNTAX_CNTR32
SNMP_SYNTAX_GAUGE32
SNMP_SYNTAX_TIMETICKS
SNMP_SYNTAX_OPAQUE
SNMP_SYNTAX_NSAPADDR
SNMP_SYNTAX_CNTR64
SNMP_SYNTAX_UINT32
SNMP_SYNTAX_NULL
SNMP_SYNTAX_NOSUCHOBJECT
SNMP_SYNTAX_NOSUCHINSTANCE
SNMP_SYNTAX_ENDOFMIBVIEW

The application must eventually call the SnmpFreeDescriptor function to enable the implementation to free resources that might have been allocated to populate the ptr members of the name and (depending upon its syntax member) value structures.

Parameters

vbl Identifies the subject varbindlist.

index Identifies the position of the subject varbind within the varbindlist.

name Points to a variable to receive the OID portion of the varbind.

value Points to a variable to receive the value portion of the varbind.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.
SNMPAPI_INDEX_INVALID	Indicates that the index parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpSetVb(3)” on page 940.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpOidCompare(3)

Purpose

Lexicographically compares two object identifiers (OIDs)

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS     SnmpOidCompare (
   IN smiLPCOID    xOID,
   IN smiLPCOID    yOID,
   IN smiUINT32    maxlen,
   OUT smiLPINT    result);
```

Description

The SnmpOidCompare function lexicographically compares two OIDs. If the maxlen value is non-zero, then its value is used as an upper limit on the number of sub-identifiers to compare. This approach is most often used to identify whether two OIDs have common prefixes or not. If the maxlen value is zero, then the len members of the two smiOID structures determine the maximum number of sub-identifiers to compare.

When the maxlen value is non-zero (but not greater than MAXOBJIDSIZE), the maximum number of sub-identifiers that are compared is the *minimum* of the maxlen input parameter and the two len members of the input OID structures. Either or both of the input OIDs can have a zero length without causing an error.

When the maxlen value is zero, the maximum number of sub-identifiers that are compared is the minimum of the two len members of the input OID structures. Either or both of the input OIDs can have a zero length without causing an error.

If the two OIDs are lexicographically equal when the maximum number of sub-identifiers are compared, then:

- If the maxlen parameter value is used as the maximum number of sub-identifiers to compare, or if the two OID parameters have equal len members which are less than the maxlen input parameter, the result value is 0 (equal).
- If an OID len member is used as the value for the maximum number of sub-identifiers to compare (because it is less than the non-zero maxlen input parameter or because the maxlen value is equal to zero), and the other OID len member value is greater, the result value is <0 or >0, depending on which OID parameter has which len value.

Parameters

xOID Points to a variable holding an object identifier to compare.

yOID Points to a variable holding an object identifier to compare.

maxlen If non-zero, indicates the number of sub-identifiers to compare. Must be less than MAXOBJIDSIZE.

result Points to a variable to receive the result of the comparison:

> 0 if xOID is greater than yOID
= 0 if xOID equals yOID

< 0 if xOID is less than yOID

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_OID_INVALID	Indicates that either or both of the input OIDs were not valid.
SNMPAPI_SIZE_INVALID	Indicates that the maxlen parameter was not valid; that is, greater than MAXOBJIDSIZE.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpOidCopy(3)

Purpose

Duplicates the specified OID

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpOidCopy (
   IN smiLPCOID   srcOID,
   OUT smiLPOID   dstOID);
```

Description

The SnmpOidCopy function copies the srcOID to the dstOID.

The member elements of the smiOID structure pointed to by the dstOID structure are ignored on input and are overwritten by the implementation upon a successful execution of this function.

The application must eventually call the SnmpFreeDescriptor function to enable the implementation to free any resources that have been allocated to populate the ptr member of the dstOID structure.

Parameters

srcOID Points to a variable holding an object identifier.

dstOID Points to a variable to receive a copy of the srcOID.

Return Values

When the function is successful, the return value is the number of sub-identifiers in the output OID. This number is also the value of the len member of the dstOID structure upon return.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_OID_INVALID	Indicates that the srcOID was not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidToStr(3)” on page 917.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpOidToStr(3)

Purpose

Converts a WinSNMP OID into a dotted numeric string

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS     SnmpOidToStr (
   IN  smiLPCOID   srcOID,
   IN smiUINT32    size,
   OUT LPSTR       string);
```

Description

The SnmpOidToStr function converts an internal representation of an OID into a dotted numeric string representation of an OID (for example, 1.2.3.4.5.6).

The format of the OID array in an smiOID structure is one integral sub-identifier per array element. That is, the string 1.3.6.1 (lstrlen=7) becomes an array of {1,3,6,1} (len=4) and vice versa.

The application should use a string buffer of MAXOBJIDSTRSIZE length for this call, to be safe. If, as is normally true, a shorter OID is actually decoded, the application can copy the resulting string to one of appropriate length and either reuse or free the space allocated to the original buffer.

A NULL-terminated string is returned for convenience. The return value, upon success, includes the terminating NULL byte.

Parameters

srcOID Points to a variable holding an object identifier to be converted.

size The size of the buffer that the application is providing to contain the string.

string Points to a buffer that receives the string that identifies the management entity.

Return Values

When the function is successful, the return value is the number of bytes, including the NULL terminating byte, in the output string. This return value may be less than or equal to the size parameter, but not greater.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_OID_INVALID	Indicates that the srcOID was not valid.
SNMPAPI_OUTPUT_TRUNCATED	Indicates that the buffer was too small.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpStrToOid(3)” on page 951.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpRecvMsg(3)

Purpose

Retrieves results of a completed SNMP request or trap for the specified session

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpRecvMsg (
   IN  HSNMP_SESSION          session,
   OUT LPHSNMP_ENTITY         srcEntity,
   OUT LPHSNMP_ENTITY         dstEntity,
   OUT LPHSNMP_CONTEXT        context
   OUT LPHSNMP_PDU            pdu);
```

Description

The SnmpRecvMsg function retrieves the results from a completed asynchronous request made on a given HSNMP_SESSION. It also receives traps registered for that session.

The implementation is only required to deliver information through SnmpRecvMsg that was contained in the incoming SNMP message it received from the transport layer. For SNMPv2, all components are included in the SNMP message itself. For SNMPv1, an implementation has several choices: it might have access to additional transport layer data and elect to use that; it can probably associate an in-bound GetResponse PDU with an out-bound request PDU and use the srcEntity and dstEntity values from that; or it can return NULL for components missing from the received SNMP message.

NetView for AIX Implementation Note

The IBM implementation attempts to provide all of the output parameters listed. However, users should check each output parameter for a NULL pointer before using.

The application is responsible for freeing the handle object resources returned by this function when it is no longer needed by the application, by calling the SnmpFreePdu, SnmpFreeEntity, and SnmpFreeContext functions when appropriate.

Note: There are four handle objects instantiated by a successful SnmpRecvMsg operation (for example, the varbindlist component of the returned PDU is *not* instantiated until called for by the application with the SnmpGetPduData function).

Replies are not necessarily received in the same order as their originating requests were sent. For traps received from SNMPv1 entities, in addition to mapping them to SNMPv2 format, the implementation must assign a non-zero RequestID. A RequestID value delivered through trap notification can possibly duplicate a RequestID used by an application on a request PDU; applications need to check for this occurrence.

When a trap is delivered by SnmpRecvMsg, it is returned in the SNMPv2 format, even if a SNMPv1 entity generated the trap. The SNMPv2 coexistence specification, as described in *RFC 1908: Coexistence between Version 1 and Version 2 of Internet-standard Network Management Framework*, specifies the mapping rules between the SNMPv1 and SNMPv2 trap formats. However, for the convenience of management applications, the final variable binding for a SNMPv1-generated trap is always

snmpTrapEnterpriseOID.0, even if the trap is a generic trap such as coldStart. For further information about mapping traps between SNMPv1 and SNMPv2, see the *NetView for AIX Programmer's Guide*.

Parameters

session Specifies the session that receives the SNMP message.

srcEntity Identifies the entity (agent role) which sent the message.

dstEntity Identifies the entity (manager role) which is to receive the message.

context Identifies the context from which the srcEntity issued the message.

pdu Identifies the PDU component of the received message.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS, and the OUT parameters are populated with their corresponding values.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. For the transport layer (TL) errors, the OUT parameters are populated with their corresponding values to enable applications to recover gracefully. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.

SNMPAPI_ALLOC_ERROR Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_SESSION_INVALID Indicates that the session parameter is not valid.

SNMPAPI_NOOP Indicates that the message queue for this session is empty.

SNMPAPI_TL_NOT_INITIALIZED
Indicates that the transport layer is not initialized.

SNMPAPI_TL_NOT_SUPPORTED
Indicates that the transport does not support protocol.

SNMPAPI_TL_NOT_AVAILABLE
Indicates that the network subsystem has failed.

SNMPAPI_TL_RESOURCE_ERROR
Indicates that a transport resource error occurred.

SNMPAPI_TL_UNDELIVERABLE
Indicates that the destination is unreachable.

SNMPAPI_TL_SRC_INVALID Indicates that the source endpoint is not valid.

SNMPAPI_TL_INVALID_PARAM
Indicates that the parameter to transport call is not valid.

SNMPAPI_TL_PDU_TOO_BIG Indicates that the PDU is too big for transport.

SNMPAPI_TL_TIMEOUT Indicates that there is no response within the timeout interval.

SNMPAPI_TL_OTHER Indicates that the transport error is undefined.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The following sections in the *NetView for AIX Programmer's Guide*:
  - transport interface support
  - asynchronous modelt
  - polling and retransmission
  - requestIDs
  - WinSNMP communications functions
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpRegister(3)

Purpose

Registers the calling application to receive or discontinue trap and inform notifications

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS   SnmpRegister (
   IN HSNMP_SESSION    session,
   IN HSNMP_ENTITY     srcEntity,
   IN HSNMP_ENTITY     dstEntity,
   IN HSNMP_CONTEXT    context,
   IN smiLPCOID        notification,
   IN smiUINT32        status);
```

Description

The SnmpRegister function registers the application's desire to receive or discontinue receiving trap and inform notifications from the specified entity of interest (dstEntity), which acts in an agent role.

Note: In WinSNMP, all traps delivered to the applications are SNMPv2 traps. If an implementation receives an SNMPv1 trap from an SNMPv1 agent, it must convert it to an SNMPv2 trap in accordance with *RFC 1908: Coexistence between Version 1 and Version 2 of Internet-standard Network Management Framework*.

For information about mapping traps between SNMPv1 and SNMPv2, see the *NetView for AIX Programmer's Guide*.

Notifications, traps or informs, are defined using OIDs, as specified in SNMPv2. Hence, an application interested in receiving coldStart traps should construct an OID corresponding to this trap based upon the *RFC 1907: Management Information Base for Version 2 of the Simple Network Management Protocol* (SNMPv2C) and use this as the notification parameter.

Note: The value of the notification parameter is used for pattern matching against the OIDs of received traps and notifications. That is, if the first 'n' sub-ids of a received SnmpTrapOID match all the sub-ids ('n') of a notification value passed to SnmpRegister, then that SnmpTrapOID is a match. Accordingly, a received SnmpTrapOID with fewer sub-ids than a given notification parameter must fail the matching process with respect to that particular notification parameter.

An application may pass NULL for any or all of the srcEntity, dstEntity, context, and notification parameters. The significance of NULL in any of these parameters is, effectively, to tell the implementation to *not* filter out any received traps or notifications on the basis of this parameter.

When the notification parameter is NULL, then the application is indicating that it is interested in registering or unregistering for any and all notifications from the dstEntity, as indicated by the status parameter.

When the status parameter contains any value other than SNMPAPI_OFF or SNMPAPI_ON, it is treated as though it were SNMPAPI_ON.

Upon receipt of a trap or notification, the hWnd parameter specified in the SnmpCreateSession call for the registered session is sent the wMsg specified. The application should call SnmpRecvMsg with this session to retrieve the appropriate results.

It is the responsibility of a Level 3 implementation to acknowledge the receipt of an InformRequest-PDU. This tells the issuing management entity that the inform made it to the implementation "platform," but not necessarily to any particular applications.

As described above, a NULL dstEntity input parameter to SnmpRegister tells the WinSNMP implementation to accept the specified notification from any and all sources. As the user application calls SnmpRecvMsg to process each notification as it arrives, SnmpRecvMsg creates a "new" srcEntity output parameter. This new entity "belongs" to the application as though it had caused its creation with SnmpStrToEntity. Put differently, the behavior in this respect is the same as for SnmpDecodeMsg. This is equally true, although less likely to occur, if a NULL srcEntity or context input parameters is passed to SnmpRegister as well.

This functionality relates to not filtering traps and notifications *received* by the implementation. It does not address the issue of how such traps and notifications are directed to the implementation in the first place. This is assumed to occur "out-of-band" from the perspective of the application making use of NULL filtering parameters as described in this section.

Parameters

session Identifies the session which is interested in registering.

srcEntity Identifies the management entity (manager role) of interest; this is the trap recipient. (This is the "source" of the notification request.)

dstEntity Identifies the management entity (agent role) of interest; this is the trap sender. (This is the "target" of the notification request.)

context Identifies the context of interest.

notification
Identifies the trap or notification OID matching sequence to be registered or un-registered.

status Indicates whether to register (SNMPAPI_ON) or un-register (SNMPAPI_OFF) for the subject notification.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session parameter is not valid.
SNMPAPI_ENTITY_INVALID	Indicates that the entity parameter is not valid.

SNMPAPI_CONTEXT_INVALID	Indicates that the context parameter is not valid.
SNMPAPI_OID_INVALID	Indicates that the notification parameter is not valid.
SNMPAPI_TL_NOT_INITIALIZED	Indicates that the transport layer is not initialized.
SNMPAPI_TL_IN_USE	Indicates that the trap port is not available.
SNMPAPI_TL_NOT_AVAILABLE	Indicates that the network subsystem has failed.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSelect(3)

Purpose

Checks the I/O status of multiple file descriptors and message queues, handling SNMP file descriptors transparently

Syntax

```
#include  <WinSNMP.h>
#include  <IBMwsnmp.h>

smiINT  SnmpSelect(
            IN smiINT nfds,
            IN fd_set *readfds,
            IN fd_set *writefds,
            IN fd_set *exceptfds,
            IN struct timeval *timeout);
```

Description

The SnmpSelect function is an IBM extension to the NetView for AIX WinSNMP API. It is similar to the Standard C Library “select” function in that it checks the file descriptors specified by the application to see if they are ready for reading (receiving) or writing (sending), or if they have an exceptional condition pending.

The application calls SnmpSelect with initialized numfds, fd_set, and timeval parameters. SnmpSelect then internally adds its own related fd information for all pending SNMP requests to the fd set specified by the application and proceeds to wait for an event. When an fd is finally ready for processing, SnmpSelect determines whether the fd is related to SNMP or whether it belongs to the application. Any SNMP related fds will be handled internally. SnmpSelect returns to the calling application when:

- An fd which the application owns is ready for processing.
- The application timeout has expired.
- A select error has occurred on one of the fds owned by the application.
- An unrecoverable SNMP error has occurred.
- There are no application-defined or SNMP-defined fds left to select upon.

Note that SnmpSelect handles all incoming SNMP data, SNMP timeout, SNMP retries, and other SNMP related errors by calling the appropriate owning session's callBack function which was specified when the session was created with SnmpCreateSession. Only errors which cannot be isolated to a specific WinSNMP session result in an SnmpSelect failure.

Parameters

nfds Specifies the number of application file descriptors to check.

readfds The set of file descriptors to be checked for reading.

writefds The set of file descriptors to be checked for writing.

exceptfds The set of file descriptors to be checked for exceptions.

timeout The application defined timeval structure.

Return Values

If the function is successful, the return value is set to the total number of file descriptors that satisfy the selection criteria.

If the time limit specified by the timeout parameter expires without an application's file descriptor being satisfied, a 0 (zero) is returned.

If the function is unsuccessful due to a select on one of the application's file descriptors, the return value is -1. An application should then check the global variable errno for the appropriate error code.

If the function is unsuccessful due to an SNMP error which cannot be associated with an owning session, the return value is -2.

Error Codes

If the return value was -1, the application should check the global variable errno for the appropriate error code. (Refer to the AIX **select** man page for further error code information.)

If the return value was -2, the application should call SnmpGetLastError or SnmpGetLastErrorStr to ascertain the reason for failure.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpSendMsg(3)” on page 927.
- See “SnmpStartup(3)” on page 942.

For further information see the following:

- The section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSendMsg(3)

Purpose

Sends an SNMP message to the specified destination entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpSendMsg (
   IN  HSNMP_SESSION   session,
   IN HSNMP_ENTITY     srcEntity,
   IN HSNMP_ENTITY     dstEntity,
   IN HSNMP_CONTEXT    context,
   IN  HSNMP_PDU       pdu);
```

Description

The SnmpSendMsg function requests that the specified PDU be transmitted to the destination entity, using the specified context and, for SNMPv2 communications, the designated source entity.

When a transmission request is received by the implementation through the SnmpSendMsg function, the WinSNMP implementation determines which version of SNMP and which transport to use. These decisions are based on the implementation's capabilities and the corresponding properties associated with the requesting session and with the remote entity which holds the context to be accessed, based on values in the local database.

This function returns immediately. If the return indicates an error, SnmpGetLastError or SnmpGetLastErrorStr should be called immediately to find out the error type. When the asynchronous request completes, the hWnd specified in the SnmpCreateSession call is sent the wMsg specified. The application should call SnmpRecvMsg with this HSNMP_SESSION to retrieve the results from the request.

Note: It is the responsibility of the WinSNMP implementation to verify the correctness of the PDU structure (and other SnmpSendMsg input parameters) and to return a failure to the caller and an extended error code through SnmpGetLastError or SnmpGetLastErrorStr. For example, for a PDU_type other than SNMP_PDU_GETBULK and SNMP_PDU_RESPONSE (if allowed), passed values (other than zero) for error_status and/or error_index would constitute a PDU structure that is not valid and the implementation should return SNMPAPI_FAILURE and set SnmpGetLastError or SnmpGetLastErrorStr to report SNMPAPI_PDU_INVALID.

An application may assign a RequestID to a PDU at any time through the SnmpCreatePdu or SnmpSetPduData functions. If the RequestID component is zero at the time of the SnmpCreatePdu call, the implementation assigns a non-zero RequestID to the PDU. An application that wants to use a zero-valued RequestID must set it to that value with the SnmpSetPduData function.

As SNMP replies do not necessarily come back in the same order as requests were sent, the application should check the RequestID of the received message to match it with the appropriate request.

If an SNMPv2 feature is requested, but the dstEntity implies an entity using SNMPv1, then the downgrading procedures defined in the *RFC 1908: Coexistence between Version 1 and Version 2 of Internet-standard Network Management Framework* are used. If downgrading is not possible (for example, an

InformRequest-PDU directed at an SNMPv1 agent), then the function is unsuccessful and SnmpGetLastError or SnmpGetLastErrorStr returns SNMPAPI_OPERATION_INVALID.

Parameters

session Identifies the session that performs the operation.

srcEntity Identifies the subject management entity.

dstEntity Identifies the target management entity.

context Identifies the target context of interest.

pdu Identifies the SNMP PDU containing the operation.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use SnmpGetLastError or SnmpGetLastErrorStr to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.

SNMPAPI_ALLOC_ERROR Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_SESSION_INVALID Indicates that a session parameter is not valid.

SNMPAPI_ENTITY_INVALID Indicates that an entity parameter is not valid.

SNMPAPI_CONTEXT_INVALID Indicates that the context parameter is not valid.

SNMPAPI_PDU_INVALID Indicates that the PDU parameter is not valid.

SNMPAPI_OPERATION_INVALID
Indicates that the PDU_type element is inappropriate for the destination entity.

SNMPAPI_TL_NOT_INITIALIZED
Indicates that the transport layer is not initialized.

SNMPAPI_TL_NOT_SUPPORTED
Indicates that transport does not support protocol.

SNMPAPI_TL_NOT_AVAILABLE
Indicates that the network subsystem has failed.

SNMPAPI_TL_RESOURCE_ERROR
Indicates that a transport resource error occurred.

SNMPAPI_TL_SRC_INVALID Indicates that the source endpoint is not valid.

SNMPAPI_TL_INVALID_PARAM
Indicates that the parameter to transport call is not valid.

SNMPAPI_TL_PDU_TOO_BIG Indicates that the PDU is too big for transport.

SNMPAPI_TL_OTHER Indicates that the transport error is undefined.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpStartup(3)” on page 942.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The following sections in the *NetView for AIX Programmer's Guide*:
  - transport interface support
  - asynchronous modelt
  - polling and retransmission
  - requestIDs
  - WinSNMP communications functions
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetPduData(3)

Purpose

Updates the specified PDU with data supplied by the calling application

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS        SnmpSetPduData (
   IN HSNMP_PDU       PDU,
   IN smiLPINT        PDU_type,
   IN smiLPINT32      request_id,
   IN smiLPINT        non_repeaters,     -- for GetBulkRequest-PDU only
   IN smiLPINT        max_repetitions,   -- for GetBulkRequest-PDU only
   IN LPHSNMP_VBL     vbl);
```

Description

The SnmpSetPduData function updates selected data elements in the specified PDU.

All parameters must be supplied to the function call, and any or all of them, except the PDU, may be NULL. No values are changed in the PDU for input parameters passed as NULL (and they are passed as pointers to values to allow for the case when NULL is the desired update value).

Not all possible combinations of individually legal component values are valid. The WinSNMP implementation must verify the validity of the PDU (and other message elements) when the application calls the SnmpSendMsg or SnmpEncodeMsg function and reject all ill-formed or otherwise illegal PDU structures.

Parameters

PDU Identifies the SNMP protocol data unit.

PDU_type
If not NULL, points to an smiINT variable that updates the PDU_type of the PDU.

request_id
If not NULL, points to an smiINT32 variable that updates the request_id of the PDU.

non_repeaters
If not NULL, points to an smiINT variable that updates the non_repeaters of the GetBulkRequest-PDU (ignored for other PDU_types).

max_repetitions
If not NULL, points to an smiINT variable that updates the max_repetitions of the GetBulkRequest-PDU (ignored for other PDU_types).

vbl If not NULL, points to an HSNMP_VBL variable that updates the handle to the varbindlist of the PDU.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_PDU_INVALID	Indicates that the PDU parameter is not valid.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.
SNMPAPI_NOOP	Indicates that all input parameters were NULL.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCreatePdu(3)” on page 867.
- See “SnmpDuplicatePdu(3)” on page 878.
- See “SnmpFreePdu(3)” on page 892.
- See “SnmpGetPduData(3)” on page 900.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP PDU functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetRetransmitMode(3)

Purpose

Sets the retransmission mode for subsequent SnmpSendMsg operations

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS     SnmpSetRetransmitMode (
   IN smiUINT32    nRetransmitMode);
```

Description

The SnmpSetRetransmitMode function enables the calling application to inform the implementation of the desired retransmission mode (for example, timeout or retry) for subsequent SnmpSendMsg operations.

SNMPAPI_ON
: Indicates that the implementation *is* doing retransmission.

SNMPAPI_OFF
: The implementation is *not* doing retransmission.

Changing the retransmission mode from SNMPAPI_OFF to SNMPAPI_ON has no effect on any SNMP communications initiated through SnmpSendMsg function calls which are outstanding prior to successful return from the subject SnmpSetRetransmitMode function call. That is, an implementation does *not* have to execute the retransmission policy for messages which it initially sent when the retransmission mode was set to SNMPAPI_OFF and to which it has not yet received a response. An implementation may elect to execute the retransmission policy on behalf of such messages in this case, but this behavior is not a requirement and applications should not count on it. The purpose of this particular specification is to enable the implementations to take maximum advantage of the SNMPAPI_OFF retransmission mode when it is in effect.

When an application changes the retransmission mode from SNMPAPI_ON to SNMPAPI_OFF, the implementation *should (but is not required to)* cancel all further retransmission attempts for any outstanding SNMP communications operations in effect prior to the call (and, of course, *must not* initiate any for subsequent SnmpSendMsg functions until the application sets the mode back to SNMPAPI_ON). Applications, however, should assume that the implementation has done so. The reason this behavior is so specified is that it might not be possible for an implementation to run through a list of outstanding SNMP communications operations and turn each one off, while also receiving new SnmpSendMsg requests and traps and notifications from prior SnmpRegister requests, without one or more previously set retransmit timers waking up. Because this may be the “critical loop” for WinSNMP implementations, we need to ensure that the implementations can handle it efficiently.

NetView for AIX Implementation Note

The IBM implementation *will* enforce the new retransmission mode for all outstanding (pending) requests. Prior to the execution of this function, the default mode is SNMPAPI_ON.

Parameters

nRetransmitMode

Sets the current retransmission mode to one of the following values:

SNMPAPI_ON
SNMPAPI_OFF

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_MODE_INVALID	Indicates that the implementation does not support the requested translation mode.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetRetry(3)

Purpose

Sets the number of retries for subsequent communication with the specified entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpSetRetry (
   IN HSNMP_ENTITY hEntity,
   IN smiUINT32   nPolicyRetry);
```

Description

The SnmpSetRetry function enables an application to set the Policy Retry count on a per destination entity basis in the implementation's local database.

The retry value is expressed as a simple unit count. If this value is zero, and the application and the implementation have agreed to SnmpSetRetransmitMode (SNMPAPI_ON), then the implementation selects an operating value for this parameter when actually executing the retransmission policy.

NetView for AIX Implementation Note

If the retry value specified is zero (0), the IBM implementation defaults the policy number of retries to 2.

Parameters

nEntity Indicates the destination entity of interest.

nPolicyRetry
Indicates the retry count for this entity to be stored in the implementation's local database.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetTimeout(3)” on page 936.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about polling and retransmission and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetTimeout(3)

Purpose

Sets the timeout value for the specific entity

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpSetTimeout (
   IN HSNMP_ENTITY hEntity,
   IN smiTIMETICKS nPolicyTimeout);
```

Description

The SnmpSetTimeout function enables an application to set the Policy Timeout value, in units of hundredths of seconds, on a per destination entity basis in the implementation's local database.

If this value is zero, and both the application and the implementation agree to SnmpSetRetransmitMode (SNMPAPI_ON), then the implementation selects an operating value for this parameter when actually executing the retransmission policy.

NetView for AIX Implementation Note

If the specified timeout value is zero (0), the IBM implementation defaults the policy timeout to 5 seconds.

Parameters

hEntity Indicates the destination entity of interest.

nPolicyTimeout
Indicates the timeout value for this entity to be stored in the implementation's local database.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_ENTITY_INVALID	Indicates that an entity parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTranslateMode(3)” on page 938.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about polling and retransmission and the section about WinSNMP local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetTranslateMode(3)

Purpose

Sets the entity/context translate mode

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpSetTranslateMode (
   IN smiUINT32   nTranslateMode);
```

Description

The SnmpSetTranslateMode function enables the calling application to inform the implementation of the desired entity/context translation mode to use for subsequent SnmpStrToEntity and SnmpStrToContext function calls:

SNMPAPI_TRANSLATED
: Indicates that the local database look-up translation mode will be used.

SNMPAPI_UNTRANSLATED_V1
: Indicates that the literal transport address and community string translation mode will be used.

SNMPAPI_UNTRANSLATED_V2
: Indicates that the literal SNMPv2 party and context IDs translation mode will be used.

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C“ Draft-Standard.

SNMPAPI_UNTRANSLATED_V2 now indicates that the literal transport address and community string translation mode is used.

Upon successful execution of the SnmpSetTranslateMode function, the requested translation mode remains in effect for all subsequent SnmpStrToEntity and SnmpStrToContext function calls until another SnmpSetTranslateMode call with a different nTranslateMode value is executed successfully.

Parameters

nTranslateMode
: Sets the current translation mode to one of the following values:

  SNMPAPI_TRANSLATED
  SNMPAPI_UNTRANSLATED_V1
  SNMPAPI_UNTRANSLATED_V2

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE and the value of SnmpGetLastError or SnmpGetLastErrorStr is set to one of the following error codes.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_MODE_INVALID	Indicates that the implementation does not support the requested translation mode.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpGetRetransmitMode(3)” on page 903.
- See “SnmpGetRetry(3)” on page 905.
- See “SnmpGetTimeout(3)” on page 907.
- See “SnmpGetTranslateMode(3)” on page 909.
- See “SnmpSetRetransmitMode(3)” on page 932.
- See “SnmpSetRetry(3)” on page 934.
- See “SnmpSetTimeout(3)” on page 936.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context translation modes and the section about local database functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpSetVb(3)

Purpose

Adds and updates varbinds in a varbindlist structure

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS     SnmpSetVb (
   IN HSNMP_VBL    vbl,
   IN smiUINT32    index,
   IN smiLPCOID    name,
   IN smiLPCVALUE  value);
```

Description

The SnmpSetVb function adds and updates varbind entries in a varbindlist.

Valid values for the index parameter range from 0 (zero) to *n*, where *n* is the total number of varbinds currently in the varbindlist as reported by the SnmpCountVbl function. An index value of 0 (zero) indicates the addition of a varbind to the varbindlist.

If the value parameter is NULL, the varbind is initialized, with the value set to NULL and with a syntax of SNMP_SYNTAX_NULL.

Parameters

vbl Identifies the target varbindlist.

index Identifies the position of the subject varbind within the varbindlist for an update operation or is zero for an add or append operation.

name Points to a variable containing the object instance name to be set.

value If not NULL, points to a variable containing the associated value to be set.

Return Values

When the function is successful, the return value is the position (index value) of the affected varbind. In the case of successful update operations, the return value equals the index parameter. In the case of add or append operations (in which the index parameter is zero), the return value is *n*+1, where *n* is the previous total count of varbinds in the varbindlist (per SnmpCountVbl).

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.

SNMPAPI_ALLOC_ERROR Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_VBL_INVALID	Indicates that the vbl parameter is not valid.
SNMPAPI_INDEX_INVALID	Indicates that the index parameter is not valid.
SNMPAPI_OID_INVALID	Indicates that the name parameter is not valid.
SNMPAPI_SYNTAX_INVALID	Indicates that the syntax field of the value parameter is not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCountVbl(3)” on page 865.
- See “SnmpCreateVbl(3)” on page 872.
- See “SnmpDeleteVb(3)” on page 876.
- See “SnmpDuplicateVbl(3)” on page 880.
- See “SnmpFreeVbl(3)” on page 894.
- See “SnmpGetVb(3)” on page 911.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP variable binding functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpStartup(3)

Purpose

Initializes and allocates the necessary resources to perform other WinSNMP functions

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpStartup (
   OUT smiLPUINT32      nMajorVersion,
   OUT smiLPUINT32      nMinorVersion,
   OUT smiLPUINT32      nLevel,
   OUT smiLPUINT32      nTranslateMode
   OUT smiLPUINT32      nRetransmitMode);
```

Description

The SnmpStartup function notifies the implementation that the calling application is going to use its services, enabling the implementation to perform any required start-up procedures and allocations and to return some useful housekeeping information to the application.

Note: Every WinSNMP application must call SnmpStartup at least once and this call must precede any other WinSNMP API function call.

For information about levels of SNMP support, see the *NetView for AIX Programmer's Guide*.

When this call is unsuccessful, the application must not make any further WinSNMP API calls, other than SnmpGetLastError or SnmpGetLastErrorStr and, if appropriate, retries to SnmpStartup. If an application calls other WinSNMP API functions without a preceding successful SnmpStartup, the implementation should return SNMPAPI_NOT_INITIALIZED.

An application which receives SNMPAPI_FAILURE and SNMP_ALLOC_ERROR in response to SnmpStartup may elect to wait and to try again later in the hope that the implementation then has adequate free resources.

SnmpStartup is idempotent. This means that an application can call it multiple times with impunity. Multiple SnmpStartup calls do not require multiple SnmpCleanup calls. Every application must call SnmpStartup at least once, before any other WinSNMP API call, and must call SnmpCleanup at least once, as the last WinSNMP API call.

Parameters

nMajorVersion
: Indicates the pointer to variable to receive the major version number of the WinSNMP API implemented.

nMinorVersion
: Indicates the pointer to variable to receive the minor version number of the WinSNMP API implemented.

nLevel
: Indicates the pointer to variable to receive the highest level of SNMP communications supported by the implementation.

nTranslateMode
Indicates the pointer to variable to receive the default entity/context translation mode in effect for the implementation.

nRetransmitMode
Indicates the pointer to variable to receive the default retransmission mode in effect for the implementation.

Return Values

When the function is successful, the return value is SNMPAPI_SUCCESS. The output parameters contain appropriate values, as follows:

nMajorVersion contains the major version number of the WinSNMP API implemented; the only legal value at this time is 1 (v1.nMinorVersion).

nMinorVersion contains the minor version number of the WinSNMP API implemented; legal values at this time are 0 (v1.0) and 1 (v1.1).

nLevel contains the highest level of SNMP communications supported by the implementation. This may be one of the following values:

SNMPAPI_NO_SUPPORT	Indicates Level 0 (Message builder)
SNMPAPI_V1_SUPPORT	Indicates Level 1 (SNMPv1 agents)
SNMPAPI_V2_SUPPORT	Indicates Level 2 (SNMPv2 agents)
SNMPAPI_M2M_SUPPORT	Indicates Level 3 (Manager-to-Manager)

nTranslateMode contains the current default mode of translation of the entity and context parameters when used as inputs to SnmpStrToEntity and SnmpStrToContext functions. This may be one of the following values:

SNMPAPI_TRANSLATED
Indicates that the local database look-up translation mode will be used.

SNMPAPI_UNTRANSLATED_V1
Indicates that the literal SNMPv1 transport address and community string translation mode will be used.

SNMPAPI_UNTRANSLATED_V2
Indicates that the literal SNMPv2 partyID and contextIDs translation mode will be used.

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C“ Draft-Standard.

SNMPAPI_UNTRANSLATED_V2 now indicates that the literal transport address and community string is used.

nRetransmitMode contains the current default retransmission mode in effect for the implementation. This may be one of the following values:

SNMPAPI_OFF	Indicates that the implementation is *not* executing the retransmission policy.

SNMPAPI_ON Indicates that the implementation *is* executing the retransmission policy.

When this function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpCleanup(3)” on page 858.
- See “SnmpClose(3)” on page 860.
- See “SnmpCreateSession(3)” on page 869.
- See “SnmpRecvMsg(3)” on page 919.
- See “SnmpRegister(3)” on page 922.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpSendMsg(3)” on page 927.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP communications functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpStrToContext(3)

Purpose

Defines a WinSNMP context identified by the input string

Syntax

```
#include  <WinSNMP.h>

HSNMP_CONTEXT    SnmpStrToContext (
   IN HSNMP_SESSION    session,
   IN smiLPCOCTETS     string);
```

Description

The SnmpStrToContext function accepts an octet string naming the collection of managed objects of interest (for SNMPAPI_TRANSLATED mode), a community string (for SNMPAPI_UNTRANSLATED_V1 mode), or a contextID (for SNMPAPI_UNTRANSLATED_V2 mode) and returns a handle to an implementation-specific representation of context information for use with the SnmpSendMsg and SnmpRegister functions.

The smiOCTETS descriptor pointed to by the string parameter in the SnmpStrToContext function is both allocated and populated by the application. Hence, SnmpFreeDescriptor should not be called to free the memory associated with the ptr member of this descriptor.

Note: Strings referenced in descriptors (such as an smiOCTETS structure) do not require a NULL terminating byte. Such a string can be used in an IN smiOCTETS parameter by setting the len member to ignore it.

When the application no longer needs this context handle, the SnmpFreeContext function should be called to release the resources associated with the handle.

When the current entity/context translation mode is SNMPAPI_TRANSLATED, the string parameter is assumed to describe a user-friendly name (in textual form) to be translated through the local database.

NetView for AIX Implementation Note

The WinSNMP local database is currently implemented as the configuration file: /usr/OV/conf/snmpv2.conf. Users may define SNMPv2C and secure SNMPv2USEC agents by inserting entries into this file.

When the current entity/context translation mode is SNMPAPI_UNTRANSLATED_V1, the string parameter is assumed to describe a literal community string (which may contain non-printable ASCII byte values).

When the current entity/context translation mode is SNMPAPI_UNTRANSLATED_V2, the string parameter is assumed to describe a literal contextID (in textual form).

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C“ Draft-Standard.

SNMPAPI_UNTRANSLATED_V2 now indicates that the string parameter is assumed to describe a literal community string. For the purpose of this StrToContext function, SNMPAPI_UNTRANSLATED_V1 and SNMPAPI_UNTRANSLATED_V2 can be used interchangeably since they both refer to an SNMP community string.

Parameters

session Identifies the handle of the allocating session.

string Identifies the pointer to an smiOCTETS descriptor identifying a collection of managed objects, community string, or contextIDs.

Return Values

When the function is successful, the return value is an HSNMP_CONTEXT handle.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.

SNMPAPI_ALLOC_ERROR Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.

SNMPAPI_OTHER_ERROR Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_SESSION_INVALID Indicates that the session handle is not valid.

SNMPAPI_CONTEXT_INVALID Indicates that the string descriptor is not valid (for example, the len or the ptr member is NULL).

SNMPAPI_CONTEXT_UNKNOWN
Indicates that the value referenced in the string descriptor is unknown.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpContextToStr(3)” on page 862.
- See “SnmpEntityToStr(3)” on page 884.
- See “SnmpFreeContext(3)” on page 886.
- See “SnmpFreeEntity(3)” on page 890.
- See “SnmpStrToEntity(3)” on page 948.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about entity and context translation modes and the section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpStrToEntity(3)

Purpose

Creates a WinSNMP entity identified by the null-terminated input string

Syntax

```
#include  <WinSNMP.h>

HSNMP_ENTITY     SnmpStrToEntity (
   IN HSNMP_SESSION     session,
   IN LPCSTR            entity);
```

Description

The SnmpStrToEntity function accepts a pointer to a null-terminated text string identifying an entity of interest and, if successful, returns a handle to an implementation-specific representation of entity information. The resulting entity handle may be used as either a srcEntity value or as a dstEntity value. The semantics of the input string are governed by the value of entity/context translation mode in effect at the time of the call.

When the application no longer needs to utilize this entity handle, the SnmpFreeEntity function should be called to release the resources associated with it.

When the current entity/context translation mode is SNMPAPI_TRANSLATED, the entity parameter is assumed to be a user-friendly textual name to be translated through the local database.

NetView for AIX Implementation Note

The WinSNMP local database is currently implemented as the configuration file: /usr/OV/conf/snmpv2.conf. Users may define SNMPv2C and secure SNMPv2USEC agents by inserting entries into this file.

When the current entity/context translation mode is SNMPAPI_UNTRANSLATED_V1, the entity parameter is assumed to be a literal transport address (in textual form). The implementation attempts to identify local database resources associated with this SNMPv1 address and supplies working defaults when no such entry exists in the local database. This is to enable "out-of-the-box" SNMPv1/UDP operation with WinSNMP.

When the current entity/context translation mode is SNMPAPI_UNTRANSLATED_V2, the entity parameter is assumed to be a literal PartyID (in textual form). The implementation attempts to identify local database resources associated with this SNMPv2 “party” and supplies working defaults when no such entry exists in the local database. This is to enable "out-of-the-box" SNMPv2/InitialPartyID operation with WinSNMP.

NetView for AIX Implementation Note

The SNMPAPI_UNTRANSLATED_V2 mode above was originally based upon the old “Party Based SNMPv2” model which is now historic. The IBM implementation now associates the SNMPAPI_UNTRANSLATED_V2 mode with the newly standardized “Community Based SNMPv2C” Draft-Standard.

SNMPAPI_UNTRANSLATED_V2 now indicates that the entity parameter is assumed to be the literal transport address (in textual form) of an SNMPv2C node. The IBM implementation attempts to identify local database resources associated with this entity and supplies working defaults when no such entry exists. If there are multiple entries in the database for this address, the first such entry is used.

Parameters

session Identifies the handle of the allocating session.

entity Identifies the pointer to a NULL-terminated text string identifying the management entity of interest.

Return Values

When the function is successful, the return value is an HSNMP_ENTITY handle.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.
SNMPAPI_SESSION_INVALID	Indicates that the session handle is not valid.
SNMPAPI_ENTITY_UNKNOWN	Indicates that the entity parameter is unknown.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpContextToStr(3)” on page 862.
- See “SnmpEntityToStr(3)” on page 884.
- See “SnmpFreeContext(3)” on page 886.
- See “SnmpFreeEntity(3)” on page 890.
- See “SnmpStrToContext(3)” on page 945.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*

- The section about entity and context translation modes and the section about entity and context functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

SnmpStrToOid(3)

Purpose

Converts a textual object identifier into an internal WinSNMP OID

Syntax

```
#include  <WinSNMP.h>

SNMPAPI_STATUS    SnmpStrToOid (
   IN  LPCSTR     string,
   OUT smiLPOID   dstOID);
```

Description

The SnmpStrToOid function converts a textual representation of the dotted numeric form of an object identifier into an internal object identifier representation.

The format of the OID array in an smiOID structure is one integral sub-identifier per array element. That is, the string 1.3.6.1 (lstrlen=7) becomes an array of {1,3,6,1} (len=4) and vice versa.

The member elements of the smiOID structure pointed to by the dstOID structure are ignored on input and are overwritten by the implementation upon a successful execution of this function.

The application must eventually call the SnmpFreeDescriptor function to enable the implementation to free any resources that might have been allocated to populate the ptr member of the dstOID structure.

This function is unsuccessful with SNMPAPI_OID_INVALID, for example, if the string input parameter is not NULL terminated, is of insufficient length, is longer than MAXOBJIDSTRSIZE, or does not constitute the textual form of a valid OID.

Parameters

string Points to a NULL terminated string to be converted.

dstOID Points to an smiOID variable to receive the converted value.

Return Values

When the function is successful, the return value is the number of sub-identifiers in the output object identifier. This number is also the value of the len member of the dstOID structure upon return.

When the function is unsuccessful, the return value is SNMPAPI_FAILURE. Use the SnmpGetLastError or SnmpGetLastErrorStr function to obtain extended error information.

Error Codes

SNMPAPI_NOT_INITIALIZED	Indicates that SnmpStartup has not successfully executed since the program began or since SnmpCleanup successfully completed.
SNMPAPI_ALLOC_ERROR	Indicates that the implementation is unable to obtain sufficient resources to carry out the requested action.
SNMPAPI_OTHER_ERROR	Indicates that an unknown, undefined, or indeterminate error occurred.

SNMPAPI_OID_INVALID Indicates that the string was not valid.

Libraries

When compiling a program that uses this function, you need to link to the following library:

- /usr/OV/lib/libnvwinsnmp.a

Related Information

- See “SnmpDecodeMsg(3)” on page 874.
- See “SnmpEncodeMsg(3)” on page 882.
- See “SnmpFreeDescriptor(3)” on page 888.
- See “SnmpGetLastError(3)” on page 896.
- See “SnmpGetLastErrorStr(3)” on page 898.
- See “SnmpOidCompare(3)” on page 913.
- See “SnmpOidCopy(3)” on page 915.
- See “SnmpOidToStr(3)” on page 917.

This function is part of the WinSNMP open interface. For further information see the following:

- The *Windows SNMP Manager API Specification Version 1.1a*
- The section about WinSNMP utility functions in the *NetView for AIX Programmer's Guide*
- The /usr/OV/conf/snmpv2.conf file for configuration information

XnvApplicationShell(3)

Purpose

Functions as the main top-level window for an application managed by the NetView for AIX graphical interface. The XnvApplicationShell widget must be used when the application is managed by the NetView for AIX graphical interface. The widget can perform Drag/Drop operations and can go inside and outside the Control Desk, a special area in the NetView for AIX server.

Syntax

```
#include "<OV/XnvApplicationShell.h>"
```

Dependencies

The XnvApplicationShell widget must be created by reference to xnvApplicationShellWidgetClass, the widget class associated with it.

Description

XnvApplicationShell is used as the main top-level window for an application that is managed by the NetView for AIX EUI. An application should have more than one XnvApplicationShell only if it implements multiple logical applications.

XnvApplicationShell inherits behavior and resources from Core, Composite Shell, WMShell, VendorShell, TopLevelShell, and ApplicationShell.

The class pointer is xnvApplicationShellWidgetClass, and the class name is xnvApplicationShell.

If the XnvApplicationShell cannot find the NetView for AIX server at the XtRealize time, the shell starts without communication with the EUI server. Also, Drag/Drop operations are disabled.

If the XnvNassociatedShell resource receives a wrong parameter, the shell behaves the same as when this resource does not exist.

The resources to set the icon representation of the shell inside the Control Desk are:

XnvNiconType
XnvNiconColor
XnvNiconPicture
XnvNiconLabel
XnvNiconLabelColor

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a value by name or by class in an .Xdefaults file, remove the XnvN or XnvC prefix and use the remaining letters. To specify one of the defined values for a resource in an .Xdefault file, remove the Xnv prefix and use the remaining letters (in either lowercase or uppercase, including any underscores between words). The codes in the access column indicate whether the given resource can be set at creation time (C), set by using XtSetValues (S), retrieved by using XtGetValues (G), or is not applicable (N/A).

Table 20. Widget Resources for XnvApplicationShell

Name	Class	Default	Type	Access
XnvNeuiManaged	XtCBoolean	False	Boolean	C
XnvNassociatedShell	XtCWidget	NULL	Widget	C
XnvNoutside	XtCBoolean	False	Boolean	C
XnvNiconType	XtCBoolean	False	Boolean	C/S/G
XnvNiconColor	XtCString	NULL	String	C/S/G
XnvNiconPicture	XtCString	NULL	String	C/S/G
XnvNiconLabel	XtCString	NULL	String	C/S/G
XnvNiconLabelColor	XtCString	NULL	String	C/S/G

The XnvApplicationShell widget replaces the Toolkit Intrinsics associated widget, ApplicationShell, and should perform the same job as the common widget. But the XnvApplicationShell widget allows the created shells to be controlled by the NetView for AIX server and to have certain characteristics, such as the Box area. This area is a small bar over the shell area that enables you to perform Drag/Drop operations. These operations move the application shells inside and outside the Control Desk, which is an application repository. In addition to the resources of its Intrinsics-associated widget (ApplicationShell), the XnvApplicationShell has the following new resources:

XnvNeuiManaged (True/False)
: Specifies whether the XnvApplicationShell will be managed by the NetView for AIX EUI. This resource, associated with the other resources, enables you to move the XnvApplicationShell to a WorkSpace called Control Desk and to drag it inside and outside at any time. If this resource is False, the other resources have no meaning.

XnvNassociatedShell (<widget_id>)
: Specifies, for secondary XnvApplicationShell widgets in the same application, the Control Desk to initially start them. This Control Desk will be the same one that uses the associated Shell referenced in XnvNassociatedShell. If XnvNoutside is set to True, XnvNassociatedShell has no meaning.

  Note: At the Realize time, if the XnvApplicationShell has the XnvNeuiManaged resource set to True, and if no NetView for AIX EUI is found, the resource is turned to False automatically and the other new resources are ignored, but no warning message appears.

XnvNoutside (True/False)
: Specifies the first place that the XnvApplicationShell widget will appear, if XnvNeuiManaged resource was set to True. If XnvNoutside is True, XnvApplicationShell is started automatically inside the NetView for AIX Control Desk. If XnvNoutside is False, XnvApplicationShell is started with no Control Desk associated with it.

XnvNiconType (XnvICON_COLOR, XnvICON_PICTURE)
: Specifies the type of the icon that will be associated with ApplicationShell to represent it inside the Control Desk.

XnvNiconColor (<color_name>)
: Specifies the color name to use as the icon background if XnvNiconType is set to True (XnvICON_COLOR).

XnvNiconPicture (<picture_filename>)
: Specifies the file name of the picture to use as the icon background if XnvNiconType is set to XnvICON_PICTURE. (XnvICON_PICTURE).

XnvNiconLabel (<label_string>)
Specifies the label name to be written in the icon.

XnvNiconLabelColor (<color_name>)
Specifies the color name to be used to paint the label color in the icon.

Note: The new resources that have only C access can be set only during the Creation time. They cannot be updated by a SetValues call.

Examples

```
Arg    args[10];
int    argcnt;
Widget shell;

argcnt = 0;
XtSetArg(args[argcnt], XnvNeuiManaged, True]; argcnt++;
XtSetArg(args[argcnt], XnvNoutside, False); argcnt++;
XtSetArg(args[argcnt], XnvNassociatedShell, NULL); argcnt++;
XtSetArg(args[argcnt], XnvNiconType, XnvICON_COLOR); argcnt++;
XtSetArg(args[argcnt], XnvNiconColor, "blue"); argcnt++;
XtSetArg(args[argcnt], XnvNiconLabel, "example"); argcnt++;
XtSetArg(args[argcnt], XnvNiconLabelColor, "black"); argcnt++;
XtSetArg(args[argcnt], XtNwidth, 400); argcnt++;
XtSetArg(args[argcnt], XtNheight, 400) argcnt++;
shell = XtAppCreateShell
        (
          "applicationName",
          "applicationClass",
          xnvApplicationShellWidgetClass,
          display,
          args,
          argcnt
        );

XtRealizeWidget(shell);
```

Note: This example creates an XnvApplicationShell managed by the NetView for AIX EUI, inside the Control Desk, with no associated shell (the shell is started in the Primary Control Desk). The icon that represents the application shell inside the Control Desk has a blue background and a black label "example." The shell will have a Box area for Drag/Drop operations and its size outside the Control Desk will be 400×400 pixels.

XnvTopLevelShell(3)

Purpose

Functions as the main top-level window for an application managed by the NetView for AIX EUI. The XnvTopLevelShell widget must be used when the application is managed by the NetView for AIX EUI. The widget can perform Drag/Drop operations and can go inside and outside the Control Desk, a special area in the NetView for AIX server.

Syntax

```
#include "<OV/XnvTopLevelShell.h>"
```

Dependencies

The XnvTopLevelShell widget must be created by reference to xnvTopLevelShellWidgetClass, the widget class associated with it.

Description

XnvTopLevelShell is used as the main top-level window for a application that is managed by the NetView for AIX EUI. An application should have more than one XnvTopLevelShell only if it implements multiple logical applications.

XnvTopLevelShell inherits behavior and resources from Core, Composite Shell, WMShell, VendorShell, and TopLevelShell.

The class pointer is xnvTopLevelShellWidgetClass, and the class name is xnvTopLevelShell.

If the XnvTopLevelShell cannot find the NetView for AIX server at the XtRealize time, the shell starts without communication with the EUI server. Also, Drag/Drop operations are disabled.

If the XnvNassociatedShell resource receives a wrong parameter, the shell behaves the same as when this resource does not exist.

The resources to set the icon representation of the shell inside the Control Desk are:

XnvNiconType
XnvNiconColor
XnvNiconPicture
XnvNiconLabel
XnvNiconLabelColor

The following table defines a set of widget resources used by the programmer to specify data. The programmer can also set the resource values for the inherited classes to set attributes for this widget. To reference a value by name or by class in an .Xdefaults file, remove the XnvN or XnvC prefix and use the remaining letters. To specify one of the defined values for a resource in an .Xdefault file, remove the Xnv prefix and use the remaining letters (in either lowercase or uppercase, including any underscores between words). The codes in the access column indicate whether the given resource can be set at creation time (C), set by using XtSetValues (S), retrieved by using XtGetValues (G), or is not applicable (N/A).

Table 21. Widget Resources for XnvTopLevelShell

Name	Class	Default	Type	Access
XnvNeuiManaged	XtCBoolean	False	Boolean	C
XnvNassociatedShell	XtCWidget	NULL	Widget	C
XnvNoutside	XtCBoolean	False	Boolean	C
XnvNiconType	XtCBoolean	False	Boolean	C/S/G
XnvNiconColor	XtCString	NULL	String	C/S/G
XnvNiconPicture	XtCString	NULL	String	C/S/G
XnvNiconLabel	XtCString	NULL	String	C/S/G
XnvNiconLabelColor	XtCString	NULL	String	C/S/G

The XnvTopLevelShell widget replaces the Toolkit Intrinsics associated widget, TopLevelShell, and should perform the same job as the common widget. But the XnvTopLevelShell widget allows the created shells to be controlled by the NetView for AIX server and to have certain characteristics, such as the Box area. This area is a small bar over the shell area that enables you to perform Drag/Drop operations. These operations move the application shells inside and outside the Control Desk, which is an application repository. In addition to the resources of its Intrinsics-associated widget (TopLevelShell), the XnvTopLevelShell has the following new resources:

XnvNeuiManaged (True/False)
: Specifies whether the XnvTopLevelShell will be managed by the NetView for AIX EUI. This resource, associated with the other resources, enables you to move the XnvTopLevelShell to a WorkSpace called Control Desk and to drag it inside and outside at any time. If this resource is False, the other resources have no meaning.

  Note: At the Realize time, if the XnvTopLevelShell has the XnvNeuiManaged resource set to True, and if no NetView for AIX EUI is found, the resource is turned to False automatically and the other new resources are ignored, but no warning message appears.

XnvNassociatedShell (<widget_id>)
: Specifies, for secondary XnvTopLevelShell widgets in the same application, the Control Desk to initially start them. This Control Desk will be the same one that uses the associated Shell referenced in XnvNassociatedShell. If XnvNoutside is set to True, XnvNassociatedShell has no meaning.

XnvNoutside (True/False)
: Specifies the first place that the XnvTopLevelShell widget will appear, if XnvNeuiManaged was set to True. If XnvNoutside is True, XnvTopLevelShell is started automatically inside the NetView for AIX Control Desk. If XnvNoutside is False, XnvTopLevelShell is started with not Control Desk associated with it.

XnvNiconType (XnvICON_COLOR, XnvICON_PICTURE)
: Specifies the type of the icon that will be associated with TopLevelShell to represent it inside the Control Desk.

XnvNiconColor (<color_name>)
: Specifies the color name to use as the icon background.

XnvNiconPicture (<picture_filename>)
: Specifies the file name of the picture to use as the icon background if XnvNiconType is set to XnvICON_PICTURE. (XnvICON_PICTURE).

XnvNiconLabel (<label_string>)
Specifies the label name to be written in the icon.

XnvNiconLabelColor (<color_name>)
Specifies the color name to be used to paint the label color in the icon.

Note: The new resources that have only C access can be set only during the Creation time. They cannot be updated by a SetValues call.

Examples

```
Arg    args[10];
int    argcnt;
Widget shell;

argcnt = 0;
XtSetArg(args[argcnt], XnvNeuiManaged, True]; argcnt++;
XtSetArg(args[argcnt], XnvNoutside, False); argcnt++;
XtSetArg(args[argcnt], XnvNassociatedShell, NULL); argcnt++;
XtSetArg(args[argcnt], XnvNiconType, XnvICON_COLOR); argcnt++;
XtSetArg(args[argcnt], XnvNiconColor, "blue"); argcnt++;
XtSetArg(args[argcnt], XnvNiconLabel, "example"); argcnt++;
XtSetArg(args[argcnt], XnvNiconLabelColor, "black"); argcnt++;
XtSetArg(args[argcnt], XtNwidth, 400); argcnt++;
XtSetArg(args[argcnt], XtNheight, 400) argcnt++;
shell = XtAppCreateShell
        (
          "applicationName",
          "applicationClass",
          xnvTopLevelShellWidgetClass,
          display,
          args,
          argcnt
        );

XtRealizeWidget(shell);
```

Note: This example creates an XnvTopLevelShell managed by the NetView for AIX EUI, inside the Control Desk, with no associated shell (the shell is started in the Primary Control Desk). The icon that represents the application shell inside the Control Desk has a blue background and a black label "example." The shell will have a Box area for Drag/Drop operations and its size outside the Control Desk will be 400×400 pixels.

Chapter 3. XOM Package

Note: The information contained in this chapter was extracted from the *X/Open & X.400 OSI Object Management API Specification, Version 2.0* (XAPIA), Chapter 6, Object Management Package.

This chapter defines the OM package. The object identifier, referred to symbolically as *om*, that is assigned to the package, as defined by this edition of this document, is that specified in ASN.1 as *{joint-iso-ccitt-mhs-motis(6) group(6) white(1) api(2) om(4)}*.

Table 22. General Information about the XOM Package

XOM Package	
Object identifier	{joint-iso-ccitt-mhs-motis(6) group(6) white(1) api(2) om(4)}
Encoding	"\x56\x06\x01\x02\x04"
Constant	OM_OM
Header file	xom.h

Class Hierarchy

This section depicts the hierarchical organization of the OM classes. Subclassification is indicated by indentation. The names of abstract classes are in *italics*. For example, Encoding is an immediate subclass of Object, an abstract class. The names of classes to which the Encode function apply are in **bold**. The Create function applies to all concrete classes.

Object

- Encoding
- **External**

Class Definitions

This section defines the OM classes.

Encoding

An instance of class **Encoding** is an object represented in a form suitable for conveyance between workspaces, transport through a network, or storage in a file. An encoding also may be a suitable way to present to an intermediate service provider (for example, a directory or message transfer system) an object it does not recognize.

This class has the attributes listed in Table 23.

Table 23. Attributes Specific to Encoding

Attribute	Value Syntax	Value Length	Value Number	Value Initially
Class	String (Object Identifier)	-	1	-
Object Class	String (Object Identifier)	-	1	-
Object Encoding	String	-	1	-
Rules	String (Object Identifier)	-	1	ber

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

If the Rules attribute is *ber* or *canonical-ber*, the syntax of the String attribute shall be String (Encoding).

Object Class Identifies the class of the object that the Object Encoding attribute encodes. The class shall be concrete.

Object Encoding The encoding itself.

Rules Identifies the set of rules that were followed to produce the Object Encoding attribute. Among the defined values of this attribute are those referred to symbolically as follows:

- **ber** Specified in ASN.1 as *{joint-iso-ccitt asn1(1) basic-encoding(1)}*, this value denotes the BER (see clause 25.2 of [2]).

Note: An instance of this class may not appear, in general, as a value whose syntax is Object(*C*), if *C* is not Encoding, even if the class of the object encoded is *C*.

External

An instance of class **External** is a data value, not necessarily describable using ASN.1, and one or more information items that describe the data value and identify its data type. This class corresponds to ASN.1's External type, and thus the class and the attributes specific to it are more fully described, indirectly, in clause 34 of [1].

This class has the attributes listed in Table 24.

Table 24. Attributes Specific to External

Attribute	Value Syntax	Value Length	Value Number	Value Initially
Class	String (Object Identifier)	-	1	-
Arbitrary Encoding	String (Bit)	-	0-1	-
ASN.1 Encoding	String (Encoding)	-	0-1	-
Data Value Descriptor	String (Object Descriptor)	-	0-1	-
Direct Reference	String (Object Identifier)	-	0-1	-
Indirect Reference	Integer	-	0-1	-
Octet Aligned Encoding	String (Octet)	-	0-1	-

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

Exactly one of the following three attributes shall be present: Arbitrary Encoding, ASN.1 Encoding, or Octet Aligned Encoding.

Arbitrary Encoding A representation of the data value as a bit string. This attribute is described more fully in clause 34 of [1].

ASN.1 Encoding The data value. This attribute may be present only if the data type is an ASN.1 type. This attribute is described more fully in clause 34 of [1].

If this attribute value's syntax is an Object syntax, the data value's implied representation is that produced by the Encode function when its Object argument is the attribute value and its Rules argument is **ber**. Thus the object's class shall be one to which the Encode function applies.

Data Value Descriptor A description of the data value. This attribute is described more fully in clause 34 of [1].

Direct Reference A direct reference to the data type. This attribute is described more fully in clause 34 of [1].

Indirect Reference An indirect reference to the data type. This attribute is described more fully in clause 34 of [1].

Octet Aligned Encoding A representation of the data value as an octet string. This attribute is described more fully in clause 34 of [1].

Object

The class *Object* represents information objects of any variety. This abstract class is distinguished by the facts that it has no superclass and that all other classes are its subclasses.

The attributes specific to this class are listed in Table 25.

Table 25. Attributes Specific to Object

Attribute	Value Syntax	Value Length	Value Number	Value Initially
Class	String (Object Identifier)	-	1	-

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

Class Identifies the object's class.

Chapter 4. XMP API Management Service Packages

This chapter describes the following three X/Open Management Protocols (XMP) API management service packages and the classes they contain:

- Common Management Service (Common) package
- CMIS Management Service (CMIS) package
- SNMP Management Service (SNMP) package

General Information

The following is general information about the XMP management service packages. This information includes the object identifier for the package, the encoding of the object identifier, the constant that represents the object identifier, and the header file that contains the constants which represent the OM classes and OM attributes in the C language.

Table 26. General Information about the Management Service Packages

Common Package	
Object identifier	{iso(1) member-national-body(2) bsi(826) disc(0) xopen(1050) XMP(1) common(1)}
Encoding	"\x2a\x86\x3a\x0\x88\x1a\x1\x01"
Constant	MP_COMMON_PKG
Header file	xmp_cmis.h
CMIS Package	
Object identifier	{iso(1) member-national-body(2) bsi(826) disc(0) xopen(1050) XMP(1) cmis(2)}
Encoding	"\x2a\x86\x3a\x0\x88\x1a\x1\x02"
Constant	MP_CMIS_PKG
Header file	xmp_xmis.h
SNMP Package	
Object identifier	{iso(1) member-national-body(2) bsi(826) disc(0) xopen(1050) XMP(1) snmp(3)}
Encoding	"\x2a\x86\x3a\x0\x88\x1a\x1\x03"
Constant	MP_SNMP_PKG
Header file	xmp_snmp.h

Common errors are defined in the Common Management Service package, while service-specific errors are defined in the CMIS Management Service package and the SNMP Management Service package, respectively.

OM Class Hierarchies

This section includes the hierarchical organization of all OM classes belonging to the three service packages (Common, CMIS, and SNMP), as well as the hierarchical organization of the OM classes specific to each of the three service packages. These hierarchies show which OM classes inherit additional OM attributes from their superclasses.

In the hierarchical lists, subclasses are indented and the names of abstract OM classes are in italics. For example, the concrete class Presentation-Address is an immediate subclass of the abstract class Address.

The following table shows all the subclasses of the OM class Object and indicates to which service package each class belongs.

Table 27 (Page 1 of 3). Hierarchical Organization of Management Service OM Classes

OM Class	Service Package
Access-Control	Common
Community-Name	Common
External-AC	Common
Action-Argument	Common
CMIS-Action-Argument	CMIS
Action-Error	CMIS
Action-Error-Info	CMIS
Action-Info	CMIS
Action-Reply	CMIS
Action-Result	Common
CMIS-Action-Result	CMIS
Action-Type-Id	CMIS
Address	Common
Network-Address	Common
AE-Title	Common
Application-Syntax	SNMP
Attribute	CMIS
Attribute-Error	CMIS
Attribute-Id	CMIS
Attribute-Id-Error	CMIS
Attribute-Id-List	CMIS
AVA	Common
Base-Managed-Object-Id	CMIS
Cancel-Get-Argument	Common
CMIS-Cancel-Get-Argument	CMIS
CMIS-Filter	CMIS
Complexity-Limitation	CMIS
Context	Common
Create-Argument	Common
CMIS-Create-Argument	CMIS
Create-Object-Instance	CMIS
Create-Result	Common
CMIS-Create-Result	CMIS
Delete-Argument	Common
CMIS-Delete-Argument	CMIS
Delete-Error	CMIS

Table 27 (Page 2 of 3). Hierarchical Organization of Management Service OM Classes

OM Class	Service Package
Delete-Result	Common
CMIS-Delete-Result	CMIS
Error	Common
Communications-Error	Common
Library-Error	Common
Service-Error	Common
CMIS-Service-Error	CMIS
SNMP-Service-Error	SNMP
System-Error	Common
Error-Info	CMIS
Event-Reply	CMIS
Event-Report-Argument	Common
CMIS-Event-Report-Argument	CMIS
SNMP-Trap-Argument	SNMP
Event-Report-Result	Common
CMIS-Event-Report-Result	CMIS
Event-Type-Id	CMIS
Filter-Item	CMIS
Get-Argument	Common
CMIS-Get-Argument	CMIS
SNMP-Get-Argument	SNMP
Get-Info-Status	CMIS
Get-List-Error	Common
CMIS-Get-List-Error	CMIS
Get-Result	Common
CMIS-Get-Result	CMIS
SNMP-Get-Result	SNMP
Invalid-Argument-Value	CMIS
Linked-Reply-Argument	Common
CMIS-Linked-Reply-Argument	CMIS
Missing-Attribute-Value	CMIS
Modification	CMIS
Multiple-Reply	CMIS
Name	Common
DS-DN	Common
No-Such-Action	CMIS
No-Such-Action-Id	CMIS
No-Such-Argument	CMIS

Table 27 (Page 3 of 3). Hierarchical Organization of Management Service OM Classes

OM Class	Service Package
No-Such-Event-Id	CMIS
No-Such-Event-Type	CMIS
Object-Class	CMIS
Object-Instance	CMIS
Object-Syntax	SNMP
Processing-Failure	CMIS
Relative-Name	Common
DS-RDN	Common
Scope	CMIS
Session	Common
Set-Argument	Common
CMIS-Set-Argument	CMIS
SNMP-Set-Argument	SNMP
Set-Info-Status	CMIS
Set-List-Error	Common
CMIS-Set-List-Error	CMIS
Set-Result	Common
CMIS-Set-Result	CMIS
SNMP-Set-Result	SNMP
Simple-Syntax	SNMP
SNMP-Response	SNMP
Specific-Error-Info	CMIS
Substring	CMIS
Substrings	CMIS
Title	Common
Entity-Name	Common
Var-Bind	SNMP

Class Restrictions: A management program is not permitted to create or modify instances of some OM classes, because these OM classes only are returned by the XMP API and never supplied to it (for example, Library-Error). The description of the OM class given in “The OM Classes” on page 969 indicates whether instances cannot be created or modified.

All OM classes can be encoded using om_encode() and om_decode(), unless stated otherwise in the class description.

The Common Management Service Package

The OM class superior to the classes in the Common package is the Object OM class. It is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*. The Common package contains the following subclasses to the OM class Object:

- *Access-Control*
  - Community-Name
  - External-AC
- *Action-Argument*
- *Action-Result*
- *Address*
  - Network-Address.
- AVA
- *Cancel-Get-Argument*
- Context
- *Create-Argument*
- *Create-Result*
- *Delete-Argument*
- *Delete-Result*
- *Error*
  - Communications-Error
  - Library-Error
  - *Service-Error*
  - System-Error.
- *Event-Report-Argument*
- *Event-Report-Result*
- Extension
- *Get-Argument*
- *Get-List-Error*
- *Get-Result*
- *Linked-Reply-Argument*
- *Name*
  - DS-DN
- *Relative-Name*
  - DS-RDN.
- Session
- *Set-Argument*
- *Set-List-Error*
- *Set-Result*
- *Title*
  - Entity-Name.

The CMIS Management Service Package

The OM class superior to the classes in the CMIS package is the Object OM class. It is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*. The CMIS package contains the following subclasses to the OM class Object.

Note: The abstract OM classes are in italics. They are defined in the Common package.

- *Action-Argument*
  - CMIS-Action-Argument.
- Action-Error
- Action-Error-Info

- Action-Info
- Action-Reply
- *Action-Result*
  - CMIS-Action-Result.
- Action-Type-Id
- Attribute
- Attribute-Error
- Attribute-Id
- Attribute-Id-Error
- Attribute-Id-List
- Base-Managed-Object-Id
- *Cancel-Get-Argument*
  - CMIS-Cancel-Get-Argument.
- CMIS-Filter
- Complexity-Limitation
- *Create-Argument*
  - CMIS-Create-Argument.
- *Create-Object-Instance*
- *Create-Result*
  - CMIS-Create-Result.
- *Delete-Argument*
  - CMIS-Delete-Argument.
- Delete-Error
- *Delete-Result*
  - CMIS-Delete-Result.
- *Error*
  - *Service-Error*
    - CMIS-Service-Error.
- Error-Info
- Event-Reply
- *Event-Report-Argument*
  - CMIS-Event-Report-Argument.
- *Event-Report-Result*
  - CMIS-Event-Report-Result.
- Event-Type-Id
- Filter-Item
- *Get-Argument*
  - CMIS-Get-Argument.
- Get-Info-Status
- *Get-List-Error*
  - CMIS-Get-List-Error.
- *Get-Result*
  - CMIS-Get-Result.
- Invalid-Argument-Value
- *Linked-Reply-Argument*
  - CMIS-Linked-Reply-Argument.
- Missing-Attribute-Value
- Modification
- Multiple-Reply
- No-Such-Action
- No-Such-Action-Id
- No-Such-Argument
- No-Such-Event-Id
- No-Such-Event-Type

- Object-Class
- Object-Instance
- Processing-Failure
- Scope
- *Set-Argument*
  - CMIS-Set-Argument.
- Set-Info-Status
- *Set-List-Error*
  - CMIS-Set-List-Error.
- *Set-Result*
  - CMIS-Set-Result.
- Specific-Error-Info
- Substring
- Substrings.

The SNMP Management Service Package

The OM class superior to the classes in the SNMP package is the Object OM class. It is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*. The SNMP package contains the following subclasses to the OM class Object.

Note: The abstract OM classes are in italics. They are defined in the Common package.

- Application-Syntax
- *Error*
  - *Service-Error*
    - SNMP-Service-Error.
- *Event-Report-Argument*
  - SNMP-Trap-Argument.
- *Get-Argument*
  - SNMP-Get-Argument.
- *Get-Result*
  - SNMP-Get-Result.
- Object-Syntax
- *Set-Argument*
  - SNMP-Set-Argument.
- *Set-Result*
  - SNMP-Set-Result.
- Simple-Syntax
- SNMP-Response
- Var-Bind.

The OM Classes

This section describes each OM class. The OM classes are presented in alphabetical order, regardless of the package to which each OM class belongs.

The OM attributes that can be found in an instance of an OM class are those that are specific to that OM class and those that are inherited from each of its superclasses. These OM attributes are defined in a table.

Note: The order in which the OM attributes are presented in the tables is not necessarily the order in which attributes are returned. That order is unknown. In general, the OM attributes are listed in the same order as the ASN.1 mapping.

Reading the OM Attribute Tables: The OM attribute tables provide the following information:

Column Name **Contents**

OM Attribute The name of each OM attribute.

Value Syntax The syntax of the values of the OM attribute. For example, Object(Object-Class) means the value of the attribute is an object and the type (class) of the object is Object-Class. Similarly, String(Printable) means the value of the attribute is a printable string.

An `any` in this column indicates that the syntax of the OM attribute is determined by the value of another OM attribute, which is indicated in the description. Use the following rules to fill in the syntax and value fields:

1. If the attribute has a syntax that is defined in one of the Common, CMIP, or SNMP packages, use that syntax and set the value accordingly.
2. If the value is one of the following ASN.1 simple types, use the appropriate OM syntax specification, with the prefix OM_S and set the value accordingly:
   - OM_S_BIT_STRING
   - OM_S_BOOLEAN
   - OM_S_GENERAL_STRING
   - OM_S_GENERALIZED_TIME
   - OM_S_GRAPHIC_STRING
   - OM_S_IA5_STRING
   - OM_S_INTEGER
   - OM_S_NULL
   - OM_S_NUMERIC_STRING
   - OM_S_OBJECT_DESCRIPTOR_STRING
   - OM_S_OBJECT_IDENTIFIER_STRING
   - OM_S_OCTET_STRING
   - OM_S_PRINTABLE_STRING
   - OM_S_TELETEX_STRING
   - OM_S_UTC_TIME_STRING
   - OM_S_VIDEOTEX_STRING
   - OM_S_VISIBLE_STRING
   - OM_S_REAL
   - OM_S_UNLIMITED_INTEGER
3. Lastly, you have the option to set the syntax field to *OM_S_ENCODING_STRING*, and use your own functions to encode (or decode) the attribute value (using BER) to (or from) a string representation. This encoded string is used as the value of the attribute.

Value Length Any restrictions on the length (in bits, octets (bytes), or characters) of each value.

Note: The value in this column is assigned a C language identifier. Refer to the *NetView for AIX Programmer's Guide* for information about the identifier.

Value Number Any restrictions on the number of values.

A `0-1` in this column indicates that the OM attribute is optional. In some cases, the OM attribute can be mutually exclusive of one or more other OM attributes. The description of the OM attribute indicates if the mutually exclusive restriction applies to it.

Notes:

1. The function calls can place additional restrictions on the number of values. Such restrictions are defined in the description of each XMP function call.
2. The value in this column is assigned a C language identifier.

Initial Value The value, if any, that the om_create() function supplies.

See For OM attributes with a value syntax of object, this column contains the number of the page that describes the OM class of the object.

Access-Control

The OM class Access-Control represents access privileges. It is information of unspecified form to be used as input to access-control functions.

It is an abstract class, which has the attributes of its superclass, Object.

Most of the CMIS functions take an access-control parameter, the value of which must be an instance of one of the subclasses of this OM class. Thus, this OM class serves to group all possible representations of access control.

There are two subclasses of this OM class:

- Community-Name, which provides SNMP access-control information
- External-AC, which provides an externally defined access control.

Action-Argument

The OM class Action-Argument represents the supplied argument of an action. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Action-Argument, which is the supplied argument of a CMIS action.

Action-Error

An instance of the OM class Action-Error is the information associated with an error for an attempted CMIS action.

An instance of this OM class has the following OM attributes.

Table 28. OM Attributes of an Action-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalized-Time)	–	0-1	–	–
action Error-Info	Object(Action-Error-Info)	–	1	–	972

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object that attempted to perform the action.

managed-Object-Instance
The instance of the managed object that attempted to perform the action.

current-Time
The time at which the response was generated.

action-Error-Info
The error information that resulted from the action requested.

Action-Error-Info

An instance of the OM class Action-Error-Info documents one action-related problem encountered while performing an action.

An instance of this OM class has the following OM attributes.

Table 29. OM Attributes of an Action-Error-Info Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
error-Status	Enum(Error-Status)	–	1	–	–
error-Info	Object(Error-Info)	–	1	–	1004

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

error-Status
The error notification for the operation. Its value is one of the following:

- *access-denied* , meaning that the requested action was not performed, due to security reasons
- *no-such-action*, meaning that the action type specified is not supported
- *no-such-argument*, meaning that the argument specified is not recognized or supported
- *invalid-argument-value*, meaning that the argument value was out of range or otherwise inappropriate.

error-Info Additional information about the error.

Action-Info

An instance of the OM class Action-Info is the information about an action that is to be performed.

An instance of this OM class has the following OM attributes.

Table 30. OM Attributes of an Action-Info Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
action Type	Object(Action-Type-Id)	–	1	–	974
action Info-Arg	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

action-Type
The action type, which indicates the action to be performed.

action-Info-Arg
Additional information, whenever it is necessary, to define further the nature, variations, or operands of the action to be performed. The syntax and semantics of this OM attribute depend upon the action requested.

The OM value syntax for this OM attribute is determined by the value of the *action-Type* OM attribute, according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

Action-Reply

An instance of the OM class Action-Reply is the information associated with the reply to a performed action.

An instance of this OM class has the following OM attributes.

Table 31. OM Attributes of an Action-Reply Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
action Type	Object(Action-Type-Id)	–	1	–	974
action Reply-Info	Any	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

action-Type
The action type, which indicates the action that was performed.

action-Reply-Info
The reply information about the action performed on a managed object. The syntax and semantics of this OM attribute depend upon the action performed. The OM value syntax for this OM attribute is determined by the value of the *action-Type* OM attribute, according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

Action-Result

The OM class Action-Result represents the result of a successful action. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Action-Result, which is the result of a successful CMIS action. It can be omitted in the last response of multiple replies.

Action-Type-Id

An instance of the OM class Action-Type-Id represents an identifier of an action.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 32. OM Attributes of an Action-Type-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
global-Form	String(Object-Identifier)	–	0-1	–	–
local-Form	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

global-Form
A registered action-type identifier.

local-Form
When this OM attribute is used, the permissible values for the integers and their meanings are defined as part of the application context in which they are used.

Address

The OM class Address represents the address of a particular management entity or method. It contains various subclasses used to define the specific location of a particular agent or manager. For example, the Network-Address subclass typically is used to define the location of an SNMP agent or manager.

It is an abstract class, which has only the OM attributes of its superclass, Object.

An address is an unambiguous name, label, or number that identifies the location of the entity or method. All addresses are represented as instances of some subclass of this OM class.

There is one subclass of this OM class:

- Network-Address (SNMP).

AE-Title

An instance of the OM class AE-Title has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 33. OM Attributes of an AE-Title

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
ae-Title-Form1	Object (DD-DN)	-	0 - 1	-	-
ae-Title-Form2	String (Object-Identifier)	-	0 - 1	-	-

Application-Syntax

Note: This OM class is applicable to SNMP only.

An instance of the OM class Application-Syntax is the data value of an application-wide type, the syntax of which corresponds to defined ASN.1 types, which are constructed on ASN.1 primitive types.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 34. OM Attributes of an Application-Syntax Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
address	Object(Network-Address)	–	0-1	–	1014
counter	Integer	–	0-1	–	–
gauge	Integer	–	0-1	–	–
ticks	Integer	–	0-1	–	–
arbitrary	String(Encoding)	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

address It represents an address from one of (possibly) several protocol families.

counter A data value of integer syntax.

It represents a nonnegative integer that monotonically increases until it reaches a maximum value; then it wraps around and starts increasing again from zero. The maximum value for counters is $2^{32}-1$ (that is, 4,294,967,295).

gauge A data value of integer syntax.

It represents a nonnegative integer, which can increase or decrease, but which latches at a maximum value. The maximum value for gauges is $2^{32}-1$ (that is, 4,294,967,295).

ticks A data value of integer syntax.

It represents a nonnegative integer that counts the time in hundredths of a second, starting from a given time period.

arbitrary A data value of string syntax.

It provides the capability to pass arbitrary ASN.1 syntax. A value is encoded, using the ASN.1 basic rules, into a string of octets. This, in turn, is encoded as an octet string, in effect double-wrapping the original ASN.1 value.

Attribute

An instance of the OM class Attribute is an attribute of a managed object.

An instance of this OM class has the following OM attributes.

Table 35. OM Attributes of an Attribute Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute-Id	Object(Attribute-Id)	–	1	–	978
attribute-Value	Any	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Id
The attribute type, which indicates the class of information provided by the attribute.

attribute-Value
The attribute value. The representation of the attribute value depends on the attribute type. The OM value syntax for this OM attribute is determined by the value of the *attribute-Id* OM attribute, according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

Attribute-Error

An instance of the OM class Attribute-Error documents one attribute-related problem encountered while performing a set operation. It is the error information generated while attempting to modify an attribute of a managed object.

An instance of this OM class has the following OM attributes.

Table 36 (Page 1 of 2). OM Attributes of an Attribute-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
error-Status	Enum(Error-Status)	–	1	–	–
modify-Operator	Enum(Modify-Operator)	–	0-1	–	–
attribute-Id	Object(Attribute-Id)	–	1	–	978

Table 36 (Page 2 of 2). OM Attributes of an Attribute-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
attribute-Value	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

error-Status

The error notification for the operation. Its value is one of the following:

- *access-denied*, meaning that the requested operation was not performed, due to security reasons.
- *no-such-attribute*, meaning that the identifier for the specified attribute is not recognized.
- *invalid-attribute-value*, meaning that the attribute value is either out of range or otherwise inappropriate.
- *invalid-operation*, meaning that the operation specified by *modify-Operator* cannot be performed on the specified attribute. (For example, the set-to-default operator is specified, but there is no defined default.)
- *invalid-operator*, meaning that the operator specified by *modify-Operator* is not recognized.

modify-Operator

Specifies the way in which the attribute was attempted to be modified. The following are the possible operators:

- *replace*, meaning that the attribute values are used to replace the current values of the attribute.
- *add-values*, meaning that the attribute values are to be added to the current values of the attribute. This operator applies only to a set-valued attribute and performs a set union (in the mathematical sense) between the current values of the attribute and the attribute values specified.
- *remove-values*, meaning that the attribute values specified are to be removed from the current values of the attribute. This operator applies only to a set-valued attribute and performs a set difference (in the mathematical sense) between the current values of the attribute and the attribute values specified. Values specified in the *attribute-Value* OM attribute that are not in the current values of the attribute do not cause an error to be returned.
- *set-to-default*, meaning one of the following:
  - When this operator is applied to a single-valued attribute, the value of the attribute is set to its default value.
  - When this operator is applied to a set-valued attribute, the values of the attribute are set to their default values, and only as many values as defined by the default are assigned.
  - When this operator is applied to an attribute group, each member of the attribute group is set to its default value.

The modify operator is present for the *invalid-operation* and *invalid-operator* errors.

attribute-Id

The attribute type, which indicates to which attribute the class of information applies.

attribute-Value
The attribute value. The representation of the attribute value depends on the attribute type. The OM value syntax for this OM attribute is determined by the value of the *attribute-Id* OM attribute, according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

The attribute value is absent for the set-to-default operator.

Attribute-Id

An instance of the OM class Attribute-Id represents an identifier of a managed-object attribute.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 37. OM Attributes of an Attribute-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
global-Form	String(Object-Identifier)	–	0-1	–	–
local-Form	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

global-Form
A registered attribute-type identifier.

local-Form
When this OM attribute is used, the permissible values and their meanings are defined as part of the application context in which they are used.

Attribute-Id-Error

An instance of the OM class Attribute-Id-Error documents one attribute-related problem encountered while performing a get operation.

An instance of this OM class has the following attributes.

Table 38. OM Attributes of an Attribute-Id-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
error-Status	Enum(Error-Status)	–	1	–	–
attribute-Id	Object(Attribute-Id)	–	1	–	978

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

error-Status
Identifies the attribute-related problem. Its value is one of the following:

- *access-denied*, meaning that the requested operation was not performed, due to security reasons
- *no-such-attribute*, meaning that the identifier for the specified attribute is not recognized.

attribute-Id
The attribute type, which identifies the attribute for which the value could not be read or modified.

Attribute-Id-List

An instance of the OM class Attribute-Id-List is a list of identifiers that specify attributes for which a value is returned.

An instance of this OM class has the following attributes.

Table 39. OM Attributes of an Attribute-Id-List Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute-Ids	Object(Attribute-Id)	–	0-128	–	978

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Ids
Specify the attributes for which values are to be returned.

AVA

An instance of the OM class AVA (attribute value assertion) provides the value of a distinguishing attribute of a managed-object instance.

An instance of this OM class has the following attributes.

Table 40. OM Attributes of an AVA Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
naming Attribute-Id	String(Object-identifier)	–	1	–	–
naming Attribute-Value	Any	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

naming-Attribute-Id
The attribute type that identifies the attribute value assertion (AVA).

naming-Attribute-Value
The attribute value. The representation of the attribute value depends on the attribute type. The OM value syntax that is allowed for this OM attribute is determined by the value of the *naming-Attribute-Id* OM attribute according to the rules expressed in "Reading the OM Attribute Tables" on page 970, and with the restrictions expressed in *ISO 10165-1, Management Information Services—Structure of Management Information Part 1: Management Information Model*.

Its syntax must be one of the following types:

- Integer
- Boolean
- Null
- Enum(*)
- String(*).

Base-Managed-Object-Id

An instance of the OM class Base-Managed-Object-Id is the pairing of the managed-object class identifier with the name of a managed-object instance.

An instance of this OM class has the following attributes.

Table 41. OM Attributes of a Base-Managed-Object-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
base-Managed Object-Class	Object(Object-Class)	–	1	–	1017
base-Managed Object-Instance	Object(Object-Instance)	–	1	–	1017

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

base-Managed-Object-Class
The class of the base managed object (a managed object used as the starting point for the selection of managed objects).

base-Managed-Object-Instance
The instance of the base managed object.

Cancel-Get-Argument

The OM class Cancel-Get-Argument represents the supplied argument of a cancel-get operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Cancel-Get-Argument, which is the supplied argument of a CMIS cancel-get operation.

CMIS-Action-Argument

An instance of the OM class CMIS-Action-Argument is the supplied argument of a CMIS action.

An instance of this OM class has the following attributes.

Table 42. OM Attributes of a CMIS-Action-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
base-Managed Object-Class	Object(Object-Class)	–	1	–	1017
base-Managed Object-Instance	Object(Object-Instance)	–	1	–	1017
access-Control	Object(Access-Control)	–	0-1	–	971
synchronization	Enum(CMIS-Sync)	–	0-1	–	–
scope	Object(Scope)	–	0-1	–	1019
filter	Object(CMIS-Filter)	–	0-1	–	987
action-Info	Object(Action-Info)	–	1	–	972

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

base-Managed-Object-Class
The class of the managed object that is to be used as the starting point for the selection of managed objects on which the filter, when supplied, is to be applied. This OM attribute is not meaningful if the corresponding *base-Managed-Object-Instance* OM attribute specifies the root of the naming tree.

base-Managed-Object-Instance
The instance of the base managed object.

access-Control
Access-control information for the purpose of obtaining permission to perform the action on the selected managed object.

synchronization
Indicates how to synchronize the selected object instances. It can take either of the following values:

- *atomic*, meaning that all managed objects selected for the operation are checked to ensure that they can perform it successfully. If at least one of the managed objects is not able to perform the operation successfully, then none perform it; otherwise, all perform it.
- *best-effort*, meaning that all managed objects selected for the operation are requested to perform it.

If this OM attribute is not supplied, best-effort synchronization is assumed. If only the base managed object is selected for the operation, this OM attribute, if present, is ignored.

scope Indicates the subtree, rooted at the base managed object, which is searched. When the scope is not specified, the scoped managed object is the specified base managed object.

filter Specifies the set of assertions that defines the filter test to be applied to the scoped managed objects. If the filter is not specified, all of the managed objects included by the scope are selected.

action-Info
Specifies the action to be performed, as well as any additional information needed to define the action further.

CMIS-Action-Result

An instance of the OM class CMIS-Action-Result is the result of a successful CMIS action.

An instance of this OM class has the following attributes.

Table 43. OM Attributes of a CMIS-Action-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalized-Time)	–	0-1	–	–
action-Reply	Object(Action-Reply)	–	0-1	–	973

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object that performed the action. This OM attribute can be absent only if the base object alone is specified as the scope.

managed-Object-Instance
The instance of the managed object that performed the action. This OM attribute can be absent only if the base object alone is specified as the scope.

current-Time
The time at which the response was generated.

action-Reply
The returned result information of the successful action.

CMIS-Cancel-Get-Argument

An instance of the OM class CMIS-Cancel-Get-Argument is the supplied argument of a CMIS cancel-get operation.

An instance of this OM class has the following attributes.

Table 44. OM Attribute of a CMIS-Cancel-Get-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
get-Invoke-Id	Integer	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

get-Invoke-Id
The identifier assigned to the previously requested and currently outstanding get operation.

CMIS-Create-Argument

An instance of the OM class CMIS-Create-Argument is the supplied argument of a CMIS create operation.

An instance of this OM class has the following attributes.

Table 45. OM Attributes of a CMIS-Create-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	1	–	1017
create Object-Instance	Object(Create-Object-Instance)	–	0-1	–	1001
access-Control	Object(Access-Control)	–	0-1	–	971
reference Object-Instance	Object(Object-Instance)	–	0-1	–	1017
attribute-List	Object(Attribute)	–	0-128	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the new managed-object instance that is to be created.

create-Object-Instance
The object-instance information that is relevant to the managed object to be created.

access-Control
Access-control information for the purpose of obtaining permission to create the specified managed object.

reference-Object-Instance
An existing managed-object instance of the same class as the managed-object instance to be created. For any attribute values, except the distinguishing attribute, not specified by the *attribute-List* OM attribute, the attribute values of the existing managed-object instance are used as default values for the new managed object.

attribute-List
Specifies the set of attribute identifiers and values to be assigned to the new managed-object instance. The remaining attributes are assigned a set of default values according to the object class definition of the new object.

CMIS-Create-Result

An instance of the OM class CMIS-Create-Result is the result of a successful CMIS create operation.

An instance of this OM class has the following attributes.

Table 46. OM Attributes of a CMIS-Create-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalized-Time)	–	0-1	–	–
attribute-List	Object(Attribute)	–	0-128	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the newly created managed object.

managed-Object-Instance
The identifier of the new managed-object instance. It must be returned if it was not supplied in the argument for the create operation.

current-Time
The time at which the response was generated.

attribute-List
The complete list of all attribute identifiers and values, if any, that were assigned to the new managed-object instance.

CMIS-Delete-Argument

An instance of the OM class CMIS-Delete-Argument is the supplied argument of a CMIS delete operation.

An instance of this OM class has the following attributes.

Table 47. OM Attributes of a CMIS-Delete-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
base-Managed Object-Class	Object(Object-Class)	–	1	–	1017
base-Managed Object-Instance	Object(Object-Instance)	–	1	–	1017
access-Control:	Object(Access-Control)	–	0-1	–	971
synchronization	Enum(CMIS-Sync)	–	0-1	–	–
scope	Object(Scope)	–	0-1	–	1019
filter	Object(CMIS-Filter)	–	0-1	–	987

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

base-Managed-Object-Class
The class of the managed object that is to be used as the starting point for the selection of managed objects on which the filter (when supplied) is to be applied. This OM attribute is not meaningful if the corresponding *base-Managed-Object-Instance* OM attribute specifies the root of the naming tree.

base-Managed-Object-Instance
The instance of the base managed object.

access-Control
Access-control information for the purpose of obtaining permission to delete the specified managed object.

synchronization
Indicates how to synchronize the selected object instances. It can take either of the following values:

- *atomic*, meaning that all managed objects selected for the operation are checked to ensure that they can perform it successfully. If at least one of the managed objects is not able to perform the operation successfully, then none perform it; otherwise, all perform it.
- *best-effort*, meaning that all managed objects selected for the operation are requested to perform it.

If this OM attribute is not supplied, best-effort synchronization is performed. If only the base managed object is selected for the operation, this OM attribute, if present, is ignored.

scope Indicates the subtree, rooted at the base managed object, that is to be searched. When the scope is not specified, the scoped managed object is the specified base managed object.

filter Specifies the set of assertions that defines the filter test to be applied to the scoped managed objects. If the filter is not specified, all of the managed objects included by the scope are selected. All the selected managed objects are to be deleted.

CMIS-Delete-Result

An instance of the OM class CMIS-Delete-Result is the result of a successful CMIS delete operation.

An instance of this OM class has the following OM attributes.

Table 48. OM Attributes of a CMIS-Delete-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalized-Time)	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object that was deleted. It can be absent only if the base object alone was specified.

managed-Object-Instance
The instance of the managed object that was deleted. It can be absent only if the base object alone was specified.

current-Time
The time at which the response was generated.

CMIS-Event-Report-Argument

An instance of the OM class CMIS-Event-Report-Argument is the supplied information about a CMIS event.

An instance of this OM class has the following OM attributes.

Table 49. OM Attributes of a CMIS-Event-Report-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	1	–	1017
managed Object-Instance	Object(Object-Instance)	–	1	–	1017
event-Time	String(Generalized-Time)	–	0-1	–	–
event-Type	Object(Event-Type-Id)	–	1	–	1006
event-Info	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object in which the event occurred.

managed-Object-Instance
The instance of the managed object in which the event occurred.

event-Time
The time at which the event was generated.

event-Type
The event type, which indicates the particular event being reported.

event-Info The information supplied with the event being reported. The syntax and semantics of this OM attribute depend upon the event reported. The OM value syntax for this OM attribute is determined by the value of the *event-Type* OM attribute, according to the rules expressed in "Reading the OM Attribute Tables" on page 970.

CMIS-Event-Report-Result

An instance of the OM class CMIS-Event-Report-Result is the result of a reported CMIS event report.

An instance of this OM class has the following attributes.

Table 50. OM Attributes of a CMIS-Event-Report-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalized-Time)	–	0-1	–	–
event-Reply	Object(Event-Reply)	–	0-1	–	1005

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object in which an event occurred.

managed-Object-Instance
The instance of the managed object that generated the event.

current-Time
The time at which the response was generated.

event-Reply
The returned result information of the reported event.

CMIS-Filter

An instance of the OM class CMIS-Filter is a set of assertions that defines the filter test to be applied to a managed object. A filter is one or more assertions about the presence or value of attributes in a managed object. A multiple assertion is an expression composed of simpler filters, referred to as nesting, using the logical operators AND, OR, and NOT. Each assertion can be a test for equality, ordering, presence, or set comparison.

Assertions about the value of an attribute are evaluated according to the matching rules associated with the attribute syntax. If an attribute value assertion is present in the filter, and that attribute is not present in the scoped managed object, then the result of the test for that attribute value assertion is evaluated as false. The managed object is selected only if the value assertion of the filter is true.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 51. OM Attributes of a CMIS-Filter Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
item	Object(Filter-Item)	–	0-1	–	1007
and	Object(CMIS-Filter)	–	0-16	–	987
or	Object(CMIS-Filter)	–	0-16	–	987
not	Object(CMIS-Filter)	–	0-1	–	987

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

item A single assertion. Each assertion relates to just one attribute of the managed object to which the filter test is to be applied.

and A collection of simpler CMIS filters. The filter is the logical conjunction of its components. The value assertion of the filter is true, unless the value assertion of any of the nested filters is false. If there are no nested components, the value assertion of the filter is true.

or A collection of simpler CMIS filters. The filter is the logical disjunction of its components. The value assertion of the filter is false, unless the value assertion of any of the nested filters is true. If there are no nested components, the value assertion of the filter is false.

not A CMIS filter. The result of this filter is reversed. The value assertion of the filter is true, if the value assertion of the enclosed filter is false. If the value assertion of the enclosed filter is true, however, then the value assertion of the filter is false.

A library error is returned by the XOM functions if an attempt is made to create a filter containing a loop—that is, a filter that contains itself, possibly through several intermediate filters. Moreover, the OM attribute *item* must be present in a nested CMIS filter in order to stop the recursion.

CMIS-Get-Argument

An instance of the OM class CMIS-Get-Argument is the supplied argument of a CMIS get operation.

An instance of this OM class has the following OM attributes.

Table 52 (Page 1 of 2). OM Attributes of a CMIS-Get-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
base-Managed Object-Class	Object(Object-Class)	–	1	–	1017
base-Managed Object-Instance	Object(Object-Instance)	–	1	–	1017
access-Control	Object(Access-Control)	–	0-1	–	971
synchronization	Enum(CMIS-Sync)	–	0-1	–	–

Table 52 (Page 2 of 2). OM Attributes of a CMIS-Get-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
scope	Object(Scope)	–	0-1	–	1019
filter	Object(CMIS-Filter)	–	0-1	–	987
attribute-Id-List	Object(Attribute-Id-List)	–	0-1	–	979

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

base-Managed-Object-Class
The class of the managed object that is to be used as the starting point for the selection of managed objects on which the filter (when supplied) is to be applied. This OM attribute is not meaningful if the corresponding *base-Managed-Object-Instance* OM attribute specifies the root of the naming tree.

base-Managed-Object-Instance
The instance of the base managed object.

access-Control
Access-control information for the purpose of obtaining permission to retrieve the attribute values from the specified managed objects.

synchronization
Indicates how to synchronize the selected object instances. It can take either of the following values:

- *atomic*, meaning that all managed objects selected for the operation are checked to ensure that they can perform it successfully. If at least one of the managed objects is not able to perform the operation successfully, then none perform it; otherwise, all perform it.
- *best-effort*, meaning that all managed objects selected for the operation are requested to perform it.

If this OM attribute is not supplied, best-effort synchronization is performed. If the base managed object alone is selected for the operation, this OM attribute, if present, is ignored.

scope Indicates the subtree, rooted at the base managed object, which is to be searched. When the scope is not specified, the scoped managed object is the specified base managed object.

filter Specifies the set of assertions that defines the filter test to be applied to the scoped managed objects. If the filter is not specified, all of the managed objects included by the scope are selected.

attribute-Id-List
A list of identifiers specifying the attributes for which values are to be returned. If the *attribute-Id-List* OM attribute is not present, all attributes are returned. If the *attribute-Id-List* OM attribute is present, but empty, no attributes are returned.

CMIS-Get-List-Error

An instance of the OM class CMIS-Get-List-Error is the result of a CMIS get operation that failed for one or more attributes.

An instance of this OM class has the following OM attributes.

Table 53. OM Attributes of a CMIS-Get-List-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalised-Time)	–	0-1	–	–
get-Info-List	Object(Get-Info-Status)	–	1-128	–	1008

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object for which one or more attributes could not be read.

managed-Object-Instance
The identifier of the managed-object instance for which one or more attributes could not be read.

current-Time
The time at which the response was generated.

get-Info-List
A list of all attribute identifiers and values that were read, as well as the identifiers and the error notification of the attributes that could not be read.

CMIS-Get-Result

An instance of the OM class CMIS-Get-Result is the result of a successful CMIS get operation.

An instance of this OM class has the following OM attributes.

Table 54. OM Attributes of a CMIS-Get-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalised-Time)	–	0-1	–	–
attribute-List	Object(Attribute)	–	0-128	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object for which zero, one, or more than one attributes were read. This OM attribute can be absent only if the base object alone was specified as the scope.

managed-Object-Instance
The identifier of the managed-object instance whose attribute values are returned. This OM attribute can be absent only if the base object alone was specified as the scope.

current-Time
The time at which the response was generated.

attribute-List
A list of all attribute identifiers and values that were read.

CMIS-Linked-Reply-Argument

An instance of the OM class CMIS-Linked-Reply-Argument is the argument of a linked reply to a requested operation.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 55. OM Attributes of a CMIS-Linked-Reply-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
get-Result	Object(CMIS-Get-Result)	–	0-1	–	990
get-List-Error	Object(CMIS-Get-List-Error)	–	0-1	–	989
set-Result	Object(CMIS-Set-Result)	–	0-1	–	996
set-List-Error	Object(CMIS-Set-List-Error)	–	0-1	–	995
action-Result	Object(CMIS-Action-Result)	–	0-1	–	982
processing-Failure	Object(Processing-Failure)	–	0-1	–	1018
delete-Result	Object(CMIS-Delete-Result)	–	0-1	–	985
action-Error	Object(Action-Error)	–	0-1	–	971
delete-Error	Object(Delete-Error)	–	0-1	–	1001

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

get-Result
A partial, successful result of a get operation.

get-List-Error
A partial result of a get operation. It contains one or more attributes that could not be read.

set-Result
A partial, successful result of a set operation.

set-List-Error
A partial result of a confirmed set operation. It contains one or more attributes that could not be modified.

action-Result
A partial, successful result of a confirmed action operation.

processing-Failure
Indicates that a general failure in processing the operation was encountered after partial results were sent.

delete-Result
A partial, successful result of a confirmed delete operation.

action-Error
A partial, negative result of a confirmed action operation.

delete-Error
A partial, negative result of a confirmed delete operation.

CMIS-Service-Error

An instance of the OM class CMIS-Service-Error reports a management error related to the provision of CMIS service.

An instance of this OM class has the following OM attributes.

Table 56. OM Attributes of a CMIS-Service-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

The problem and parameter attributes are inherited from the OM class Error, which is defined in "Error" on page 1004.

class Identifies the class of the object.

problem Provides details of a CMIS-Service-Error. Refer to Table 57 for a list and description of the standard values for this OM attribute.

parameter
Provides additional information about the error. However, some error notifications do not provide any additional information. In this case, the OM attribute *parameter* is absent.

The syntax for *parameter* depends on the value of the *problem* OM attribute. The following table lists the possible values and meanings for *problem* and the corresponding meanings and syntax for *parameter*.

Table 57 (Page 1 of 3). Problem and Parameter Values for a CMIS-Service-Error Object

Value and Meaning of Problem	Syntax and Meaning of Parameter
access-denied	Absent
The requested operation was not performed, due to security reasons.	No additional information is provided.
class-instance-conflict	Object(Base-Managed-Object-Id)
The managed-object instance is not a member of the specified class.	Identifies the managed object.

Table 57 (Page 2 of 3). Problem and Parameter Values for a CMIS-Service-Error Object

Value and Meaning of Problem	Syntax and Meaning of Parameter
complexity-limitation	Object(Complexity-Limitation)
The requested operation was not performed because an OM attribute (scope, filter, or synchronization) was too complex.	Contains the OM attribute that was too complex. **Note:** This OM attribute can be absent.
duplicate-managed-object-instance	Object(Object-Instance)
The new managed-object instance value supplied by the invoker of the create operation was already registered for a managed object of the specified class.	Specifies the name that was already registered.
get-list-error	Object(CMIS-Get-List-Error)
One or more attribute values were not read.	Contains the attributes that could not be read, as well as those that were read.
invalid-argument-value	Object(Invalid-Argument-Value)
The event argument value or action argument value was out of range or otherwise inappropriate.	Contains the event type or the action type and the argument value that is not valid.
invalid-attribute-value	Object(Attribute)
The attribute value was out of range or otherwise inappropriate.	Contains the attribute type and the attribute value that is not valid.
invalid-filter	Object(CMIS-Filter)
Contains the assertion that is not valid or the unrecognized logical operator.	Specifies the filter expression that is not valid.
invalid-object-instance	Object(Object-Instance)
The name of the object instance does not comply with the naming rules.	Contains the name that is not valid for the managed-object instance.
invalid-scope	Object(Scope)
The scope value is not valid.	Contains the scope value that is not valid.
missing-attribute-value	Object(Missing-Attribute-Value)
A required attribute value was not supplied and a default value was not available.	Identifies the attributes for which some values were required but were not supplied.
mistyped-operation	Absent
The invoke identifier of the get operation does not refer to a get operation.	No additional information is provided.
no-such-action	Object(No-Such-Action)
The action type is not recognized.	Specifies the action type.
no-such-argument	Object(No-Such-Argument)
The event or action is not recognized.	Contains the event type or the action type. May also contain the object class to which the event or the action is related.
no-such-attribute	Object(Attribute-Id)
The identifier for an attribute or attribute group is not recognized.	Contains the unrecognized attribute identifier.
no-such-event-type	Object(No-Such-Event-Type)
The event is not recognized.	Contains the event type and the object class to which the event refers.

Table 57 (Page 3 of 3). Problem and Parameter Values for a CMIS-Service-Error Object

Value and Meaning of Problem	Syntax and Meaning of Parameter
no-such-invoke-id	Integer
The invoke identifier of the get operation is not recognized.	Specifies the invoke identifier that is not valid for the get operation.
no-such-object-class	Object(Object-Class)
The class of the managed object is not recognized.	Contains the managed-object class identifier.
no-such-object-instance	Object(Object-Instance)
The instance of the managed object is not recognized.	Contains the name of the managed-object instance.
no-such-reference-object	Object(Object-Instance)
The reference-object instance is not recognized.	Contains the name of the referenced managed-object instance.
operation-cancelled	Absent
The get operation was canceled by a cancel-get operation, and no further attribute values will be returned by this invocation of the get service.	No additional information is provided.
processing-failure	Object(Processing-Failure)
A general failure was encountered during processing of the operation.	Indicates a specific error and specific information about the error. **Note:** This OM attribute can be absent.
set-list-error	Object(CMIS-Set-List-Error)
One or more attribute values were not modified.	Specifies a set of attributes, the values of those that were modified, and the values and the modify operator of those that could not be changed.
synchronization-not-supported	Enum(CMIS-Sync)
The type of synchronization is not supported.	Specifies the type of synchronization that is not supported.

CMIS-Set-Argument

An instance of the OM class CMIS-Set-Argument is the supplied argument of a CMIS set operation.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 58 (Page 1 of 2). OM Attributes of a CMIS-Set-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
base-Managed Object-Class	Object(Object-Class)	–	1	–	1017
base-Managed Object-Instance	Object(Object-Instance)	–	1	–	1017
access-Control	Object(Access-Control)	–	0-1	–	971
synchronization	Enum(CMIS-Sync)	–	0-1	–	–
scope	Object(Scope)	–	0-1	–	1019

Table 58 (Page 2 of 2). OM Attributes of a CMIS-Set-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
filter	Object(CMIS-Filter)	–	0-1	–	987
modification List	Object(Modification)	–	1-128	–	1012

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

base-Managed-Object-Class
The class of the managed object that is to be used as the starting point for the selection of managed objects on which the filter (when supplied) is to be applied. This OM attribute is not meaningful if the corresponding *base-Managed-Object-Instance* OM attribute specifies the root of the naming tree.

base-Managed-Object-Instance
The instance of the base managed object.

access-Control
Access-control information for the purpose of obtaining permission to modify the attribute values of the specified managed objects.

synchronization
Indicates how to synchronize the selected object instances. It can take either of the following values:

- *atomic*, meaning that all managed objects selected for the operation are checked to ensure that they can perform it successfully. If at least one of the managed objects is not able to perform the operation successfully, then none perform it; otherwise, all perform it.
- *best-effort*, meaning that all managed objects selected for the operation are requested to perform it.

If this parameter is not supplied, best-effort synchronization is performed. If the base managed object alone is selected for the operation, this OM attribute, if present, is ignored.

scope Indicates the subtree, rooted at the base managed object, that is to be searched. When the scope is not specified, the scoped managed object is the specified base managed object.

filter Specifies the set of assertions that defines the filter test to be applied to the scoped managed object. If the filter is not specified, all of the managed objects included by the scope are selected.

modification-List
A list that specifies the attribute identifier, the modify operator, and the attribute value to be set. The attribute value can be absent in the set-to-default operation.

CMIS-Set-List-Error

An instance of the OM class CMIS-Set-List-Error is the result of a CMIS set operation that failed for one or more attributes.

An instance of this OM class has the following OM attributes.

Table 59. OM Attributes of a CMIS-Set-List-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalised-Time)	–	0-1	–	–
set-Info-List	Object(Set-Info-Status)	–	1-128	–	1022

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object for which one or more attributes could not be modified.

managed-Object-Instance
The identifier of the managed-object instance for which one or more attributes could not be modified.

current-Time
The time at which the response was generated.

set-Info-List
The list of all attribute identifiers and values that were modified, together with the identifiers and the error notification of the attributes that could not be changed.

CMIS-Set-Result

An instance of the OM class CMIS-Set-Result is the result of a successful CMIS set operation.

An instance of this OM class has the following OM attributes.

Table 60. OM Attributes of a CMIS-Set-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalised-Time)	–	0-1	–	–
attribute-List	Object(Attribute)	–	0-128	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object for which zero, one, or more than one attributes were modified. This OM attribute can be absent only if the base object alone was specified as the scope.

managed-Object-Instance
The identifier of the managed-object instance for which the attribute values were modified. This OM attribute can be absent only if the base object alone was specified as the scope.

current-Time
The time at which the response was generated.

attribute-List
A list of all attribute identifiers and values that were modified.

Communications-Error

An instance of the OM class Communications-Error reports an error occurring in the other services supporting the Management Information Services.

A management program is not permitted to create or modify instances of this OM class.

An instance of this OM class has the following OM attributes.

Table 61. OM Attributes of a Communications-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Integer	–	0-1	–	–

Note: This attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

The problem and parameter attributes are inherited from the OM class Error, which is defined in "Error" on page 1004.

class Identifies the class of the object.

problem Its value is *communications-problem*. The involved session is abruptly terminated (no longer usable). No more results are returned for the outstanding operations or notifications.

parameter
The syntax value of this OM attribute is integer.

The communications error occurs only when the Communication Infrastructure cannot send the message to the proper recipient. The probable causes of a communication error are listed as follows:

- the recipient (agent/manager) is not running
- the agent has not been registered in the ORS

Community-Name

An instance of the OM class Community-Name represents access privileges of Internet. It is information of unspecified form to be used as input to access-control functions.

An instance of this OM class has the following OM attributes.

Table 62. OM Attributes of a Community-Name Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
community	String(Octet)	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

community
The name of the community to which the management program belongs. This name is used as input to the authentication service for SNMP.

Complexity-Limitation

An instance of the OM class Complexity-Limitation is the information that describes a complexity limitation error.

An instance of this OM class has the following OM attributes.

Table 63. OM Attributes of a Complexity-Limitation Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
scope	Object(Scope)	–	0-1	–	1019
filter	Object(CMIS-Filter)	–	0-1	–	987
synchronization	Enum(CMIS-Sync)	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

scope Indicates that the complexity limitation is related to the specified scope.

filter Indicates that the complexity limitation is related to the specified filter assertions.

synchronization
Indicates that the complexity limitation is related to the synchronization specified for the managed objects. It can take either of the following values:

- *atomic*, meaning that all managed objects selected for the operation are checked to ensure that they can perform it successfully. If at least one of the managed objects is not able to perform the operation successfully, then none perform it; otherwise, all perform it.
- *best-effort*, meaning that all managed objects selected for the operation are requested to perform it.

Context

An instance of the OM class Context defines the characteristics of a management interaction. These characteristics are specific to a particular operation but are often used unchanged for many operations.

An instance of this OM class has the following OM attributes. The OM attributes of the Context OM class are divided into two functional groups: (1) service controls, and (2) local controls.

Table 64. OM Attributes of a Context Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
Service Controls					
access-Control	Object(Access-Control)	–	0-1	–	971
mode	Enum(Mode)	–	1	con-firmed	–
priority	Enum(Priority)	–	1	medium	–
Local Controls					
asynchronous	Boolean	–	1	false	–
size-Limit	Integer	–	0-1	–	–
time-Limit	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

Although the presumption is that changes to the context will be made infrequently, the values can be changed after every operation, if required. Each argument is represented by one of the OM attributes of the Context OM class.

Service Controls

class Identifies the class of the object.

access-Control
The value for access control that is used as input to the authentication service.

mode Indicates that the management CMIS service is requested in a confirmed or nonconfirmed mode. It is meaningful only for the mp_set_req(), mp_event_report_req(), and mp_action_req() functions. Its value is either the constant MP_T_CONFIRMED, or MP_T_NON_CONFIRMED.

priority The priority, relative to other management service requests, according to which the management information service provider attempts to satisfy the request. This is not a guaranteed service. This OM attribute is without effect if it is not supported by the MIS provider. Its value must be one of the following:

- *low*
- *medium*
- *high*.

Local Controls

asynchronous
Indicates whether the XMP API operates in asynchronous mode.

This OM attribute is applicable only for those functions that can be called in asynchronous mode as well as in synchronous mode.

The value is one of the following:

- *false*, meaning that the operation is to be performed sequentially (in synchronous mode); the management program is blocked until a result or error is returned.
- *true*, meaning that the operation is to be performed in asynchronous mode (nonblocking). The management program can perform multiple, concurrent asynchronous operations and can associate a result obtained from mp_receive() with the original operation. The maximum number of outstanding concurrent operations can be specified in the configuration file. The default value of the constant MP_MAX_OUTSTANDING_OPERATIONS is 2000.

size-Limit Applicable only to functions that are used in synchronous mode and that can have linked replies.

If the *size-Limit* attribute is present and its value is less than zero, the attribute is ignored. If the *size-Limit* attribute is present and its value is greater than zero, the value indicates the maximum number of linked responses about which mp_get_req(), mp_set_req(), mp_action_req(), or mp_delete_req() return information. If this limit is exceeded, the service is abandoned for any remaining replies, and a Library-Error (*size-limit-exceeded*) is returned. In this case, the *parameter* attribute of the Library-Error has the syntax object(Multiple-Reply). The number of received, partial results is equal to the *size-Limit* value. The objects that are returned are unspecified (since they can depend, for example, on the timing of interactions between management programs).

time-Limit
Applicable only to functions that are used in synchronous mode.

If the *time-Limit* attribute is present and its value is less than zero, the attribute is ignored. If the *time-Limit* attribute is present and its value is greater than zero, the value indicates the maximum elapsed time, in seconds, within which the service must be provided. (It does *not* indicate the processing time devoted to the request.) If this limit is reached, the service is abandoned for any remaining linked replies, and a Library-Error (*time-limit-exceeded*) is returned. If partial results were received, the *parameter* attribute of the Library-Error has the syntax object(Multiple-Reply) and contains the partial results. If partial results were not received, the *parameter* attribute is absent.

Management programs can assume that an object of the OM class Context, created with default values of all its OM attributes, will work with all the XMP functions. You can use the constant MP_DEFAULT_CONTEXT as an argument to the XMP functions, instead of creating an OM object with default values.

Create-Argument

An instance of the OM class Create-Argument is the supplied argument of a create operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Create-Argument, which is the supplied argument of a CMIS create operation.

Create-Object-Instance

An instance of the OM class Create-Object-Instance is the object instance information provided for a new managed object.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 65. OM Attributes of a Create-Object-Instance Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
superior Object-Instance	Object(Object-Instance)	–	0-1	–	1017

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Instance
The instance of the managed object that is to be registered.

superior-Object-Instance
An existing managed-object instance that is the superior of the new managed-object instance.

Create-Result

An instance of the OM class Create-Result represents the result of a successful create operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Create-Result, which is the result of a successful CMIS create operation.

Delete-Argument

The OM class Delete-Argument represents the supplied argument of a delete operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Delete-Argument is the supplied argument of a CMIS delete operation.

Delete-Error

An instance of the OM class Delete-Error documents a problem encountered while performing a delete operation.

An instance of this OM class has the following OM attributes.

Table 66. OM Attributes of a Delete-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
current-Time	String(Generalised-Time)	–	0-1	–	–
delete-Error-Info	Enum(Delete-Error-Info)	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object for which deletion was attempted.

managed-Object-Instance
The instance of the managed object for which deletion was attempted.

current-Time
The time at which the response was generated.

delete-Error-Info
The error notification for the operation. Its value can be only the following:

access-denied, meaning that the requested delete operation was not performed, due to security reasons.

Delete-Result

The OM class Delete-Result represents the result of a successful delete operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Delete-Result, which is the result of a successful CMIS delete operation.

DS-DN

An instance of the OM class DS-DN represents the name of a managed object.

An instance of this OM class has the following OM attributes.

Table 67. OM Attributes of a DS-DN Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
RDNs	Object(DS-RDN)	–	0-16	–	1003

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

RDNs For a managed object, defines the path through the naming tree from its root to the managed object. The order of the values is important:

1. The first value is closest to the root.
2. The last value is the relative distinguished name of the object.

Note: The DS-DN of the root is the null name (with no *RDNs* values).

DS-RDN

An instance of the OM class DS-RDN is a relative distinguished name (RDN). An RDN uniquely identifies the immediate subordinate of a managed object in the naming tree.

An instance of this OM class has the following OM attributes.

Table 68. OM Attributes of a DS-RDN Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
AVAs	Object(AVA)	–	1	–	979

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

AVAs An attribute value assertion (AVA) is composed of an attribute type and an attribute value. The AVA attribute types used for an RDN can be a management attribute type selected for naming managed objects. This OM attribute represents the attribute value assertions that are defined in a name-binding as the single component of the RDN of the managed object.

Entity-Name

An instance of the OM class Entity-Name represents the name of a management program.

An instance of this OM class has the following OM attributes.

Table 69. OM Attributes of an Entity-Name Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
entity	String(Printable)	251	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

entity A management program name or a system name.

Error

The OM class Error comprises the parameters common to all errors.

There are four subclasses of this OM class:

- Communications-Error
- Library-Error
- Service-Error
- System-Error.

Since each subclass represents a particular type of error, details about a particular error are returned in an instance of the appropriate subclass.
The OM class Error has the following OM attributes:

Table 70. OM Attributes of an Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

problem Provides details about the error. Each *problem* value is listed and described under the error OM class to which it belongs.

parameter

Provides additional information about the error. The syntax of *parameter* depends on the value of the *problem* OM attribute.

Some error notifications do not provide any additional information. In this case, the OM attribute *parameter* is absent.

Error-Info

An instance of the OM class Error-Info provides additional information for an Action-Error-Info.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 71. OM Attributes of an Error-Info Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
action-Type	Object(Action-Type-Id)	–	0-1	–	974
action-Argument	Object(No-Such-Argument)	–	0-1	–	1015
argument-Value	Object(Invalid-Argument-Value)	–	0-1	–	1009

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

action-Type
The action type that indicates which action was attempted.

action-Argument
The action type and, optionally, the corresponding managed-object class for which the argument type was not valid.

argument-Value
The action type and, optionally, the argument value that were inappropriate.

Event-Reply

An instance of the OM class Event-Reply is the reply to an event report.

An instance of this OM class has the following OM attributes.

Table 72. OM Attributes of an Event-Reply Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
event-Type	Object(Event-Type-Id)	–	1	–	1006
event-Reply-Info	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

event-Type
The event type, which indicates the particular event being reported.

event-Reply-Info
The reply information for the event report. The syntax and meaning of this OM attribute depend upon the event reported. The OM value syntax for this OM attribute is determined by the value of the *event-Type* OM attribute according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

Event-Report-Argument

The OM class Event-Report-Argument represents the supplied argument about an event. It is an abstract class, which has the attributes of its superclass, Object.

There are two subclasses of this OM class:

- CMIS-Event-Report-Argument, which is the supplied information about a CMIS event
- SNMP-Trap-Argument, which provides the information conveyed in an SNMP trap.

Event-Report-Result

The OM class Event-Report-Result represents the result of an event report. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Event-Report-Result, which is the result of a CMIS event report.

Event-Type-Id

An instance of the OM class Event-Type-Id represents an identifier of an event report.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 73. OM Attributes of an Event-Type-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
global-Form	String(Object-Identifier)	–	0-1	–	–
local-Form	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

global-Form
A registered event-type identifier.

local-Form
When this OM attribute is used, the permissible values for the integers and their meanings are defined as part of either the application context or the package in which they are used.

External-AC

An instance of the OM class External-AC represents an externally defined access-control parameter. It is information of unspecified form to be used as input to access-control functions.

An instance of this OM class has the following OM attributes.

Table 74. OM Attributes of an External-AC Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
external-AC	Object(External)	–	1	–	‡

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

‡: The external-AC class is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

external-AC
The externally defined access-control attribute.

Filter-Item

An instance of the OM class Filter-Item is a component of a CMIS-Filter object. It is an assertion about the existence or value of a single attribute type in a managed object.

The value of the filter item is undefined if:

- The *attribute-Id* is not known.
- The *attribute-Value* does not conform to the attribute syntax defined for that attribute identifier.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 75. OM Attributes of a Filter-Item Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
equality	Object(Attribute)	–	0-1	–	976
substrings	Object(Substrings)	–	0-8	–	1029
greater-Or-Equal	Object(Attribute)	–	0-1	–	976
less-Or-Equal	Object(Attribute)	–	0-1	–	976
present	Object(Attribute-Id)	–	0-1	–	978
subset-Of	Object(Attribute)	–	0-1	–	976
superset-Of	Object(Attribute)	–	0-1	–	976
non-Null-Set Intersection	Object(Attribute)	–	0-1	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

- *class*, which identifies the class of the object.
- *equality*, meaning that the value assertion for the filter item is true if the managed object contains an attribute of the specified type— the attribute value of which is equal to that being asserted (according to the equality matching rule in force). Otherwise, the value assertion for the filter item is false.
- *substrings*, meaning that the value assertion for the filter item is true if the managed object contains an attribute of the specified attribute type— the value of which contains all of the specified substrings in the given order. (Those specified values depend on the attribute type.) Otherwise, the value assertion for the filter item is false.
- *greater-Or-Equal*, meaning that the value assertion for the filter item is true if, and only if, the managed object contains an attribute of the specified type, and the asserted value is greater than or equal to the attribute value (using the appropriate ordering algorithm).

- *less-Or-Equal*, meaning that the value assertion for the filter item is true if, and only if, the managed object contains an attribute of the specified type, and the asserted value is less than or equal to the attribute value (using the appropriate ordering algorithm).
- *present*, meaning that the value assertion for the filter item is true if the managed object contains an attribute of the specified type. Otherwise, the value assertion for the filter item is false.
- *subset-Of*, meaning that the value assertion for the filter item is true if, and only if, the managed object contains a set-valued attribute of the specified type, and the asserted value is a subset (in the mathematical sense) of the attribute value.
- *superset-Of*, meaning that the value assertion for the filter item is true if, and only if, the managed object contains a set-valued attribute of the specified type, and the asserted value is a superset (in the mathematical sense) of the attribute value.
- *non-Null-Set-Intersection*, meaning that the value assertion for the filter item is true if, and only if, the managed object contains a set-valued attribute of the specified type, and the intersection of the asserted value with the attribute value is not empty.

Get-Argument

The OM class Get-Argument represents the supplied argument of a get operation. It is an abstract class, which has the attributes of its superclass, Object.

There are two subclasses of this OM class:

- CMIS-Get-Argument, which is the supplied argument of a CMIS get operation
- SNMP-Get-Argument, which is the supplied argument of an SNMP get operation.

Get-Info-Status

An instance of the OM class Get-Info-Status is a component of the returned attribute list in a get operation.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 76. OM Attributes of a Get-Info-Status Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute Id-Error	Object(Attribute-Id-Error)	–	0-1	–	978
attribute	Object(Attribute)	–	0-1	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Id-Error
The attribute type and the error notification of an attribute that could not be read. The possible error notification is either *access-denied* or *no-such-attribute*.

attribute The type and value of the attribute that was read.

Get-List-Error

The OM class Get-List-Error represents the result of a get operation that failed for one or more attributes. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Get-List-Error, which is the result of a CMIS get operation that failed for one or more attributes.

Get-Result

The OM class Get-Result represents the result of a successful get operation. It is an abstract class, which has the attributes of its superclass, Object.

There are two subclasses of this OM class:

- CMIS-Get-Result, which is the result of a successful CMIS get operation
- SNMP-Get-Result, which is the result of a successful SNMP get or get-next operation.

Invalid-Argument-Value

An instance of the OM class Invalid-Argument-Value is the information associated with an *invalid-argument-value* error notification.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 77. OM Attributes of an Invalid-Argument-Value Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
action-Value	Object(Action-Info)	–	0-1	–	972
event-Value	Object(Event-Reply)	–	0-1	–	1005

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

action-Value
The action type and, if it exists, the action argument value that is not valid that could not be performed.

event-Value
The event type and, if it exists, the event information value that is not valid for the event reported.

Library-Error

An instance of the OM class Library-Error reports an error detected by the XMP function library.

An application is not permitted to create or modify instances of this OM class.

Each function has several possible errors that can be detected by the library itself and that are returned directly by the subroutine. These errors occur when the library itself is unable to perform an action, submit a service request, or parse a response from the system management service.

An instance of this OM class has the following OM attributes.

Table 78. OM Attributes of a Library-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

The problem and parameter attributes are inherited from the OM class Error, which is defined in "Error" on page 1004.

class Identifies the class of the object.

problem Identifies the particular library error that has occurred. The error code section for each XMP function lists only those errors that can be returned by the particular function.

parameter

Provides additional information about the Library-Error. Its OM value syntax is determined by the value of the OM attribute *problem*. Some of the errors do not include any additional information; in this case, the OM attribute *parameter* is absent.

The following list gives the possible cause of the failure and its associated information:

- *bad-address*, meaning that an address that is not valid was supplied.
- *bad-argument*, meaning that an argument that is not valid was supplied.
- *bad-class*, meaning that the OM class of either an argument, result, linked-reply, or error is not supported for this operation.
- *bad-context*, meaning that a context argument that is not valid was supplied.
- *bad-error*, meaning that a service error that is not valid was supplied.
- *bad-linked-reply*, meaning that a linked reply that is not valid was supplied.
- *bad-procedural-use*, meaning that the procedural use of linked replies does not comply with the ISO and X/Open standards, or that the permitted service primitive chaining is violated.
- *bad-result*, meaning that a result that is not valid was supplied.
- *bad-session*, meaning that a session that is not valid was supplied.
- *bad-title*, meaning that a title that is not valid was supplied.
- *bad-workspace*, meaning that a workspace argument that is not valid was supplied.

- *miscellaneous*, meaning that a miscellaneous error occurred during interaction with the system management service. This error is returned if the XMP API cannot clear a transient system error by retrying the affected system call.
- *no-such-operation*, meaning that the library has no knowledge of the designated operation or notification in progress, or that the response does not match the invoked operation or notification.
- *not-supported*, meaning that an attempt was made to use an option that is not yet available, or that was not agreed upon for use on the session.
- *session-terminated*, meaning that the session is terminated, and that the results of an outstanding operation are no longer available.
- *size-limit-exceeded*, meaning that the maximum number of linked responses about which the requested service should return information has been reached. The *parameter* OM attribute specifies a Multiple-Reply object, which contains a number of received partial results that is equal to the size-limit value. The syntax of *parameter* is object(Multiple-Reply).
- *time-limit-exceeded*, meaning that the maximum elapsed time within which the requested service must be provided has been reached. The *parameter* OM attribute specifies a Multiple-Reply object, which contains any received, partial results. Its syntax is object(Multiple-Reply). This OM attribute can be absent.
- *too-many-operations*, meaning that no more management operations can be performed until at least one asynchronous operation has been completed.
- *too-many-sessions*, meaning that no more management sessions can be started.

Linked-Reply-Argument

The OM class Linked-Reply-Argument is the argument of a linked reply to a requested management operation. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Linked-Reply-Argument, which is the supplied argument of a linked reply to a CMIS management operation.

Missing-Attribute-Value

An instance of the OM class Missing-Attribute-Value represents a list of attribute identifiers for which values are missing. This OM class is applicable only to a create operation.

An instance of this OM class has the following OM attributes.

Table 79. OM Attributes of a Missing-Attribute-Value Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
set-Of Attribute-Id	Object(Attribute-Id)	–	1-128	–	978

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

set-Of-Attribute-Id
A list of attribute-type identifiers.

Modification

An instance of the OM class Modification specifies how to modify an attribute.

An instance of this OM class has the following OM attributes.

Table 80. OM Attributes of a Modification Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute-Id	Object(Attribute-Id)	–	1	–	978
attribute-Value	Any	–	0-1	–	–
modify-Operator	Enum(Modify-Operator)	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Id
Either the attribute type, which indicates the class of information given by this attribute, or the identifier of an attribute group. An attribute group identifier can be specified only when the modify operator is set-to-default.

attribute-Value
The attribute values to be used in the modification. This OM attribute must be present if the modify operator is *not* set-to-default.

The representation of the attribute value depends on the attribute type. The OM value syntax for this OM attribute is determined by the value of the *attribute-Id* OM attribute according to the rules expressed in “Reading the OM Attribute Tables” on page 970.

modify-Operator
Specifies the way in which the attribute values, if supplied, are to be applied to the attribute. The possible operators are the following:

- *replace*, meaning that the attribute values are used to replace the current values of the attribute.
- *add-values*, meaning that the attribute values are to be added to the current values of the attribute. This operator applies only to a set-valued attribute and performs a set union (in the mathematical sense) between the current values of the attribute and the attribute values specified.
- *remove-values*, meaning that the attribute values specified are to be removed from the current values of the attribute. This operator applies only to a set-valued attribute and performs a set difference (in the mathematical sense) between the current values of the attribute and the attribute values specified. Values specified in the *attribute-Value* OM attribute that are not in the current values of the attribute do not cause an error to be returned.

- *set-to-default*, meaning one of the following:
  - When this operator is applied to a single-valued attribute, the value of the attribute is set to its default value.
  - When this operator is applied to a set-valued attribute, the values of the attribute are set to their default values, and only as many values as defined by the default are assigned.
  - When this operator is applied to an attribute group, each member of the attribute group is set to its default value.

The OM attribute *modify-Operator* is optional. If it is not specified, the replace operator is assumed.

Multiple-Reply

An instance of the OM class Multiple-Reply is the completed result of a successful get, set, action, or delete operation performed in synchronous, confirmed mode.

Multiple replies to a single operation can only occur if the invoker selects multiple managed objects, or requests an action on a single managed object in which the action is defined to produce multiple responses.

An instance of this OM class has the following OM attributes.

Table 81. OM Attributes of a Multiple-Reply Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
replies	Object(CMIS-Linked-Reply-Argument)	–	1 or more	–	991

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

replies The results of a get, set, action, or delete operation.

Name

The OM class Name represents the name of an entry in a local registration file (LRF).

A name unambiguously distinguishes the managed object from all other objects whose entries appear in the Object Registration Service (ORS). A name is a distinguished name. It is unique, since no other distinguished name identifies the same object.

It is an abstract class, which has only the attributes of its superclass, Object.

This OM class serves to group all possible representations of names for managed objects. It is used to construct an instance of the OM class Session.

There is one subclass of this OM class:

DS-DN, which provides a representation for names, including distinguished names.

Network-Address

An instance of the OM class Network-Address represents an address from the Internet protocol family.

An instance of this OM class has the following OM attributes.

Table 82. OM Attributes of a Network-Address Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
ip-Address	String(Octet)	4	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

ip-Address
Represents a 32-bit Internet address. It is represented as an octet string of length 4, in network byte-order.

No-Such-Action

An instance of the OM class No-Such-Action is the information associated with a *no-such-action* error notification.

An instance of this OM class has the following OM attributes.

Table 83. OM Attributes of a No-Such-Action Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	1	–	1017
action-Type	Object(Action-Type-Id)	–	1	–	974

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
Identifies the managed-object class to which the action is related.

action-Type
The action type, which indicates the particular action.

No-Such-Action-Id

An instance of the OM class No-Such-Action-Id is an alternative to the information associated with a *no-such-argument* error notification.

An instance of this OM class has the following OM attributes.

Table 84. OM Attributes of a No-Such-Action-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
action-Type	Object(Action-Type-Id)	–	1	–	974

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
Identifies the managed-object class to which the action is related.

action-Type
The action type, which indicates the particular action.

No-Such-Argument

An instance of the OM class No-Such-Argument represents the information associated with a *no-such-argument* error.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 85. OM Attributes of a No-Such-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
action-Id	Object(No-Such-Action-Id)	–	0-1	–	1015
event-Id	Object(No-Such-Event-Id)	–	0-1	–	1016

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

action-Id The action type and, optionally, the managed-object class identifier of the action.

event-Id The event type and, optionally, the managed-object class to which the event is related.

No-Such-Event-Id

An instance of the OM class No-Such-Event-Id is an alternative to the information associated with a *no-such-argument* error notification.

An instance of this OM class has the following OM attributes.

Table 86. OM Attributes of a No-Such-Event-Id Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	0-1	–	1017
event-Type	Object(Event-Type-Id)	–	1	–	1006

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
Identifies the managed-object class to which the event is related.

event-Type
The event type, which indicates the particular event reported.

No-Such-Event-Type

An instance of the OM class No-Such-Event-Type is the information associated with a *no-such-event-type* error notification.

An instance of this OM class has the following OM attributes.

Table 87. OM Attributes of a No-Such-Event-Type Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	1	–	1017
event-Type	Object(Event-Type-Id)	–	1	–	1006

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
Identifies the managed-object class to which the event is related.

event-Type
The event type, which indicates the particular event reported.

Object-Class

An instance of the OM class Object-Class represents an identifier of a managed-object class.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 88. OM Attributes of an Object-Class Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
global-Form	String(Object-Identifier)	–	0-1	–	–
local-Form	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

global-Form
A registered object class identifier.

local-Form
When this OM attribute is used, the permissible values for the integers and their meanings are defined as part of the application context in which they are used.

Object-Instance

An instance of the OM class Object-Instance is the name of a managed-object instance.

An instance of this OM class can have the following OM attributes.

Table 89. OM Attributes of an Object-Instance Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
local-DN	Object(DS-DN)	–	1	–	1002

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

distinguished-Name
The sequence of relative distinguished names that define the path through the naming tree from its root to the managed object. The distinguished name of the root of the naming tree is the null name (no DS-RDN values). The order of the values is important:

1. The first value is closest to the root.
2. The last value is the relative distinguished name (RDN) of the object.

This name has two parts:

- The initial part is interpreted as a system identifier relative to the global root of the naming structure. It identifies the system managed object to which the operation is directed.
- The final part is interpreted in relation to this system managed object and identifies a managed-object instance within the system.

The initial part of the name can be used as input to the LRF or to the ORS to get the Entity-Name and the address of the agent that is in charge of the managed-object instance.

Object-Syntax

Note: This OM class is applicable only to SNMP.

An instance of the OM class Object-Syntax is the data value of any object type, the syntax of which corresponds to a simple type, an Application-Wide type, or a building set or sequence of those types.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 90. OM Attributes of an Object-Syntax Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
simple	Object(Simple-Syntax)	–	0–1	–	1023
application Wide	Object(Application-Syntax)	–	0–1	–	975

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

simple A data value of simple type.

application-Wide
A data value of application-wide syntax.

Processing-Failure

An instance of the OM class Processing-Failure indicates an error that occurred during the processing of a CMIS operation.

An instance of this OM class has the following OM attributes.

Table 91. OM Attributes of a Processing-Failure Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
managed Object-Class	Object(Object-Class)	–	1	–	1017
managed Object-Instance	Object(Object-Instance)	–	0-1	–	1017
specific Error-Info	Object(Specific-Error-Info)	–	1	–	1028

Note: The attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

managed-Object-Class
The class of the managed object to which the error is related.

managed-Object-Instance
The instance of the managed object to which the error is related. This OM attribute must be present if Processing-Failure is a linked reply.

specific-Error-Info
Information about the error.

Relative-Name

The OM class Relative-Name represents the name of a managed object in the naming tree.

It is an abstract class, which has only the attributes of its superclass, Object.

A relative distinguished name (RDN) is a part of a name and only distinguishes the object from others, which are called its siblings. This OM class serves to group all possible representations of RDNs. An OM attribute value that is an RDN is an instance of the subclass of this OM class.

There is one subclass of this OM class:

DS-RDN, which provides a representation for relative distinguished names.

Scope

An instance of the OM class Scope indicates the subtree, rooted at the base managed object, that is to be searched. The default scope is the base object alone.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 92. OM Attributes of a Scope Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
named-Numbers	Enum(Scope)	–	0-1	–	–
individual-Levels	Integer	–	0-1	–	–
base-To-Nth-Level	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

named-Numbers
Indicates the standard scope levels. Its value is one of the following:

- *base-object*, meaning the base object alone

- *first-level-only*, meaning the first-level subordinates of the base object
- *whole-subtree*, meaning the base object and all of its subordinates.

individual-Levels
A positive integer indicating the level to be selected.

base-To-Nth-Level
A positive integer indicating the depth (that is, the range of levels from 0 (zero) through the integer value) that is to be selected. The base object and all of its subordinates, including the Nth level, are selected.

For *individual-Levels* and *base-To-Nth-Level*, a value of 0 (zero) has the same meaning as a value of *base-object* for *named-Numbers*.

For *individual-Levels*, a value of 1 has the same meaning as a value of *first-level-only* for *named-Numbers*.

Service-Error

The OM class Service-Error reports a management error related to the provision of service.

There are two subclasses of this OM class:

- CMIS-Service-Error, which provides service errors related to the use of CMIS
- SNMP-Service-Error, which provides service errors related to the use of SNMP.

It is an abstract OM class, which has only the OM attributes of its superclasses, which are Object and Error.

Session

An instance of the OM class Session identifies a particular link from the management program to the communications infrastructure. The OM class contains all the information that describes a particular management interaction. The parameters that control such a session are set up in an instance of this OM class, which is then passed as an argument to mp_bind(). This sets the OM attributes that describe the actual characteristics of the session and starts the session. A session that has been started in this manner must be passed as the first argument to each XMP function. No attributes of a started session can be changed.

The function mp_unbind() is used to terminate the session, after which the parameters can be modified and a new session started using the same instance, if required. Multiple concurrent sessions can be run by using multiple instances of this OM class.

A session allows a requesting management program (the requester) to exchange information with another management program (the responder).

A session thus enables:

- A manager to access the MIB. The responding agent resolution is processed by the management information service provider, according to the managed objects accessed.
- An agent to receive indications and report notifications to the possible recipient managers.

An instance of this OM class has the following OM attributes.

Table 93. OM Attributes of a Session Object

OM Attributes	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
requestor Title	Object(Title)	–	0-1	–	1031
role	Integer	–	0-1	see below	–
file-Descriptor	Integer	–	1	see below	–
access-Control	Object(Access-Control)	–	0-1	–	971
functional-Units	Integer	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

requestor-Title
Indicates the distinguished name of the user of this session (the requester). It is the system name of the requesting management program.

The OM attribute *requestor-Title* is required for management programs that need to process indicates messages using the mp_receive() function call. This is the normal case for agents, and for managers that are expected to process incoming event reports.

role Indicates the roles acted by the requester. The value is specified by ORing zero, one, or more of the following values:

- *managing*, invoker of management operations
- *monitoring*, performer of management notifications
- *performing*, performer of management operations
- *reporting*, invoker of management notifications.

The manager role corresponds to the values *managing*, *monitoring*, or both, while the agent role corresponds to the values *performing*, *reporting*, or both.

Both roles are assumed to be acted if this OM attribute is not specified.

file-Descriptor
Indicates the file descriptor associated with the session. The value is used by the mp_wait() function call. Its use for any other purpose is unspecified.

If the session is not started, the value is MP_NO_VALID_FILE_DESCRIPTOR.

access-Control
Privileged information to be used by access-control functions for the establishment of default access rights for all exchanges on the session. Subsequent exchanges can specify additional access-control information. This information is used by access-control functions, in conjunction with the default access privileges, to determine the access status of the initiator of the session for this and subsequent exchanges.

functional-Units
It identifies the service primitives and parameters supported by the session. Its value is specified by ORing zero, one, or more of the following values:

- *fu-multiple-object-selection*

- *fu-filter*
- *fu-multiple-reply*
- *fu-cancel-get*.

All these functional units are assumed to be in use if this OM attribute is not supplied.

A default session can be created by passing the constant MP_DEFAULT_SESSION as the *session* parameter to mp_bind().

The default session does not include a *requestor-Title* OM attribute; therefore, unsolicited messages cannot be received.

Set-Argument

The OM class Set-Argument represents the supplied argument of a set operation. It is an abstract class, which has the attributes of its superclass, Object.

There are two subclasses of this OM class:

- CMIS-Set-Argument, which is the supplied argument of a CMIS set operation
- SNMP-Set-Argument, which is the supplied argument of an SNMP set operation.

Set-Info-Status

An instance of the OM class Set-Info-Status is an element of the returned attribute list in a set operation.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 94. OM Attributes of a Set-Info-Status Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute Error	Object(Attribute-Error)	–	0-1	–	976
attribute	Object(Attribute)	–	0-1	–	976

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Error
The attribute type and the error notification of an attribute that could not be modified.

attribute The type and value of an attribute that was modified.

Set-List-Error

The OM class Set-List-Error represents the result of a set operation that failed for one or more attributes. It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

CMIS-Set-List-Error, which is the result of a CMIS set operation that failed for one or more attributes.

Set-Result

The OM class Set-Result represents the result of a successful set operation. It is an abstract class, which has the attributes of its superclass, Object.

There are two subclasses of this OM class:

- CMIS-Set-Result, which is the result of a successful CMIS set operation
- SNMP-Set-Result, which is the result of a successful SNMP set operation.

Simple-Syntax

Note: This OM class is applicable only to SNMP.

An instance of the OM class Simple-Syntax is the data value of a simple type, the syntax of which corresponds to every ASN.1 primitive type. Only the ASN.1 primitive types integer, octet string, object identifier and NULL are permitted. These are sometimes referred to as nonaggregate types.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 95. OM Attributes of a Simple-Syntax Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
number	Integer	–	0-1	–	–
string	String(Octet)	–	0-1	–	–
object	String(Object-Identifier)	–	0-1	–	–
empty	NULL	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

number A data value of integer syntax.

If an enumerated integer is listed as an object type, then a named number having the value 0 (zero) cannot be present in the list of enumerations. Use of this value is prohibited.

string A data value of string syntax.

object A data value of object-identifier syntax.

empty No data value syntax.

SNMP-Get-Argument

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Get-Argument is the supplied argument of an SNMP get operation.

An instance of this OM class has the following OM attributes.

Table 96. OM Attributes of an SNMP-Get-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
responder Ip-Address	Object(Network-Address)	–	1	–	1014
var-Id-List	String(Object-Identifier)	–	1 or more	–	–
access-Control	Object(Access-Control)	–	0-1	–	997

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

responder-Ip-Address
Represents the responder address.

var-Id-List A simple list of variable names.

access-Control
Represents access-control information. The only subclass of the OM class Access-Control that is permitted is Community-Name.

SNMP-Get-Result

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Get-Result is the information returned as a positive response to a requested SNMP operation. It is generated upon receipt of the indication of a get or get-next operation.

An instance of this OM class has the following OM attributes.

Table 97. OM Attributes of an SNMP-Get-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
var-Bind-List	Object(Var-Bind)	–	1 or more	–	1031

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

var-Bind-List
A simple list of the variable names and corresponding values that were read.

SNMP-Response

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Response is the information returned as a negative response to a requested SNMP operation. It is generated upon receipt of the indication of a get, get-next, or set operation.

An instance of this OM class has the following OM attributes.

Table 98. OM Attributes of an SNMP-Response Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
error-Index	Integer	–	0-1	–	–
var-Bind-List	Object(Var-Bind)	–	1 or more	–	1031

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

error-Index
The index of the object name component of the first object that is not valid that was encountered in the *var-Bind-List* OM attribute.

var-Bind-List
A simple list of variable names and corresponding values.

SNMP-Service-Error

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Service-Error reports a management error related to the provision of SNMP service.

An instance of this OM class has the following OM attributes.

Table 99. OM Attributes of an SNMP-Service-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Object(SNMP-Response)	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

The problem and parameter attributes are inherited from the OM class Error, which is defined in “Error” on page 1004.

class Identifies the class of the object.

problem This OM attribute provides details of the SNMP-Service-Error. The following are possible causes of failure:

- *too-big*, meaning that the operation argument exceeds a local limitation of the performer.
- *no-such-name*, meaning that a specified object name is unknown or unavailable for the requested operation, or is an aggregate type.
- *bad-value*, meaning that a specified object value is not appropriate.
- *read-only*, meaning that the value of a named object cannot be modified.
- *gen-err*, meaning that the value of a named object cannot be retrieved.

parameter
This OM attribute provides additional information about the error. Its syntax is object(SNMP-Response). This OM attribute contains a simple list of variable names and corresponding values, plus the index of the object name component of the first faulty object encountered in that variable bindings list.

SNMP-Set-Argument

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Set-Argument is the information supplied as the argument of a requested SNMP set operation.

An instance of this OM class has the following OM attributes.

Table 100. OM Attributes of an SNMP-Set-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
responder Ip-Address	Object(Network-Address)	–	1	–	1014
var-Bind-List	Object(Var-Bind)	–	1 or more	–	1031
access-Control	Object(Access-Control)	–	0-1	–	971

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

responder-Ip-Address
It represents a 32-bit Internet address.

var-Bind-List
A simple list of variable names and corresponding values.

access-Control
It represents access-control information. The only subclass of the OM class Access-Control that is permitted is Community-Name.

SNMP-Set-Result

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Set-Result is the information returned in response to a requested SNMP operation. It is generated upon receipt of the indication of a set operation.

An instance of this OM class has the following OM attributes.

Table 101. OM Attributes of an SNMP-Set-Result Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
var-Bind-List	Object(Var-Bind)	–	1 or more	–	1031

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

var-Bind-List
A simple list of the variable names and corresponding values that were modified.

SNMP-Trap-Argument

Note: This OM class is applicable only to SNMP.

An instance of the OM class SNMP-Trap-Argument is the information conveyed in an SNMP trap.

An instance of this OM class has the following OM attributes.

Table 102. OM Attributes of an SNMP-Trap-Argument Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
responder Ip-Address	Object(Network-Address)	–	1	–	1014
enterprise	String(Object-Identifier)	–	1	–	–
agent-Addr	Object(Network-Address)	–	1	–	1014
generic-Trap	Enum(Generic-Trap)	–	1	–	–
specific-Trap	Integer	–	1	–	–
time-Stamp	Integer	–	1	–	–
var-Bind-List	Object(Var-Bind)	–	0 or more	–	1031

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

responder-Ip-Address
Represents a 32-bit Internet address.

enterprise Represents the type of object generating the trap.

agent-Addr
Represents the address of the object generating the trap.

generic-Trap
Represents the type of generic trap. Its value is one of the following:

- *cold-start*, meaning that disturbing reinitialization is in progress.
- *warm-start*, meaning that graceful reinitialization is in progress.
- *link-down*, meaning that a failure in communication links has been detected.
- *link-up*, meaning that a communication link has been established.
- *authentication-failure*, meaning that a received message violates security rules.
- *egp-neighbor-loss*, meaning that a neighbor of the Exterior Gateway Protocol (EGP) has been affected.
- *enterprise-specific*, meaning that some enterprise-specific event has occurred.

specific-Trap
A specific code, present even if the value of *generic-Trap* is not *enterprise-specific*.

time-Stamp
The time elapsed between the last initialization or reinitialization of the network entity and the generation of the trap.

var-Bind-List
A simple list of variable names and corresponding values.

Specific-Error-Info

An instance of the OM class Specific-Error-Info is a single-error type and its associated information.

An instance of this OM class has the following OM attributes.

Table 103. OM Attributes of a Specific-Error-Info Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
error-Id	String(Object-Identifier)	–	1	–	–
error-Info	Any	–	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

error-Id Indicates a particular error.

error-Info Additional information, when necessary, to define further the nature of the error. The syntax and semantics of this OM attribute depend upon the error. The OM value syntax for this OM attribute is determined by the value of the *error-Id* OM attribute, according to the following rules:

1. The first possibility is that the error type and the representation of the corresponding values are defined in a package, such as the selected error types that are defined in the Management Content packages in Chapter 5, “XMP API Management Contents Packages.” In this case, error values are represented as specified.
2. The second possibility is that the error type is not known, but the value is an ASN.1 simple type, such as an integer or a string. In this case, the value is represented in the corresponding type specified in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.
3. The last possibility is that the error type is not known and the value is an ASN.1 structured type. In this case, the value is represented in Basic Encoding Rules (BER) with the OM string syntax.

In cases 1 and 2, the XMP API provides an automatic encode and decode functionality.

Where error values have OM syntax string(*), they can be long, segmented strings. The XOM functions om_read() and om_write() should be used to access them.

Substring

An instance of the OM class Substring identifies the string involved in a Filter-Item for which the assertion is *substrings*.

An instance of this OM class has the following OM attributes.

Table 104. OM Attributes of a Substring Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
attribute-Id	Object(Attribute-Id)	–	1	–	978
string	String(*)	1 or more	1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

attribute-Id
The attribute type, which indicates the class of information given by this attribute.

string The attribute values. The representation of the attribute value (the syntax of which is noted as string(*)) is determined by the attribute type.

Substrings

An instance of the OM class Substrings identifies the string involved in a Filter-Item for which the assertion is *substrings*.

An instance of this OM class can have the following OM attributes.

Note: In addition to the *class* attribute, an instance *must have one, and only one,* of the other OM attributes.

Table 105. OM Attributes of a Substrings Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
initial-String	Object(Substring)	–	0-1	–	1029
Any-String	Object(Substring)	–	0-1	–	1029
final-String	Object(Substring)	–	0-1	–	1029

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

initial-String
If present, the substring that is to match the initial portion of the attribute value.

any-String
If present, a set of substrings, each of them matching a portion of the attribute value.

final-String
If present, the substring that is to match the final portion of the attribute value.

System-Error

An instance of the OM class reports an error occurring in the underlying operating system.

A management program is not permitted to create or modify instances of this OM class.

An instance of this OM class has the following OM attributes.

Table 106. OM Attributes of a System-Error Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
problem	Enum(Problem)	–	1	–	–
parameter	Any	–	0-1	–	–

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

The problem and parameter attributes are inherited from the OM class Error, which is defined in "Error" on page 1004.

class Identifies the class of the object.

problem Identifies the cause of the failure. Its value is the same as that of *errno*, which is defined in the C language.

parameter

Provides additional information about the error. Its syntax value is integer. Each value corresponds to a specific error, which is defined in the C language.

Title

The OM class Title contains one subclass that is used to define the specific management program or system name responsible for a managed-object instance.

It is an abstract class, which has the attributes of its superclass, Object.

There is one subclass of this OM class:

Entity-Name, which provides either the name or the location of a management program.

Var-Bind

Note: This OM class is applicable only to SNMP.

An instance of the OM class Var-Bind is the pairing of the name and the value of a variable. The term *variable* refers to an instance of a managed object.

An instance of this OM class has the following OM attributes.

Table 107. OM Attributes of a Var-Bind Object

OM Attribute	Value Syntax	Value Length	Value Number	Initial Value	See
class	String(Object-Identifier)	–	1	–	–
name	String(Object-Identifier)	–	1	–	–
value	Object(Object-Syntax)	–	1	–	1018

Note: The class attribute is inherited from the OM class Object, which is defined in the *X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification*.

class Identifies the class of the object.

name Identifies the object type.

value The value of the designated object.

Chapter 5. XMP API Management Contents Packages

This chapter presents the Management Contents Packages provided by this XMP implementation. As stated in the NetView for AIX Programmer's Guide, these packages represent the specific management information, and describe how the values of the attributes are mapped to XOM classes. The following packages are provided:

LNV Package Contains the attributes that need to be used when an XMP application talks to an existing NetView for AIX CMOT agent such as the Event Sieve Manager.

DMI Package provides means to create the XOM classes that map the values of the attributes of the managed object classes defined in ISO 10165-2, Management Information Services—Structure of Management Information Part 2: Definition of Management Information.

LNV Package Object Identifier

The *#define* constants related to this package are in the include file */usr/OV/include/lnv.h*.

The LNV Package is assigned the OSI Object Identifier {1.3.18.0.0.3315.68}.

Object Identifier Table for LNV Attributes

Table 108. Object Identifiers for LNV Attributes

Attribute	Object Identifier
A-CMOT-System-Id	{ 1 3 6 1 2 1 9 3}

Information Syntax Table for LNV Attribute Value

Table 109. Information Syntax for LNV Attribute Value

Attribute	Attribute Syntax	Multi-Valued
A-CMOT-System-Id	Object (CMOT-System-Id)	No

OM Attribute Table for CMOT-System-Id

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 110. OM Attributes of a CMOT-System-Id

OM Attribute	Value Syntax	Value Length	Value Number
Inet-Addr	String(Octet)	-	0 or 1

DMI Package Object Identifier

The SMI-PART2 (DMI) package is assigned the OSI Object Identifier { iso(1) member-national-body(2) bsi(826) disc(0) xopen(1050) xmp(1) dmi(4) }.

Object Identifier Tables

This section contains object identifier tables for the DMI package.

Object Identifiers for DMI Object Classes

Table 111. Object Identifiers for DMI Object Classes

Object Class	Object Identifier
O-Alarm-Record	{ 2 9 3 2 3 1 }
O-Attribute-Value-Change-Record	{ 2 9 3 2 3 2 }
O-Discriminator	{ 2 9 3 2 3 3 }
O-Event-Forwarding-Discriminator	{ 2 9 3 2 3 4 }
O-Event-Log-Record	{ 2 9 3 2 3 5 }
O-Log	{ 2 9 3 2 3 6 }
O-Log-Record	{ 2 9 3 2 3 7 }
O-Object-Creation-Record	{ 2 9 3 2 3 8 }
O-Object-Deletion-Record	{ 2 9 3 2 3 9 }
O-Relationship-Change-Record	{ 2 9 3 2 3 10 }
O-Security-Alarm-Report-Record	{ 2 9 3 2 3 11 }
O-State-Change-Record	{ 2 9 3 2 3 12 }
O-System	{ 2 9 3 2 3 13 }
O-Top	{ 2 9 3 2 3 14 }

Object Identifiers for DMI Attributes

Table 112 (Page 1 of 4). Object Identifiers for DMI Attributes

Attribute	Object Identifier
A-Active-Destination	{ 2 9 3 2 7 49 }
A-Additional-Information	{ 2 9 3 2 7 6 }
A-Additional-Text	{ 2 9 3 2 7 7 }
A-Administrative-State	{ 2 9 3 2 7 31 }
A-Alarm-Status	{ 2 9 3 2 7 32 }
A-Allomorphs	{ 2 9 3 2 7 50 }
A-Attribute-Identifier-List	{ 2 9 3 2 7 8 }
A-Attribute-List	{ 2 9 3 2 7 9 }
A-Attribute-Value-Change-Definition	{ 2 9 3 2 7 10 }
A-Availability-Status	{ 2 9 3 2 7 33 }
A-Back-Up-Destination-List	{ 2 9 3 2 7 51 }
A-Back-Up-Object	{ 2 9 3 2 7 40 }
A-Backed-Up-Object	{ 2 9 3 2 7 41 }
A-Backed-Up-Status	{ 2 9 3 2 7 11 }
A-Capacity-Alarm-Threshold	{ 2 9 3 2 7 52 }

Table 112 (Page 2 of 4). Object Identifiers for DMI Attributes

Attribute	Object Identifier
A-Confirmed-Mode	{ 2 9 3 2 7 53 }
A-Control-Status	{ 2 9 3 2 7 34 }
A-Correlated-Notifications	{ 2 9 3 2 7 12 }
A-Corrupted-PD-Us-Received-Counter	{ 2 9 3 2 7 72 }
A-Corrupted-PD-Us-Received-Threshold	{ 2 9 3 2 7 89 }
A-Current-Log-Size	{ 2 9 3 2 7 54 }
A-Destination	{ 2 9 3 2 7 55 }
A-Discriminator-Construct	{ 2 9 3 2 7 56 }
A-Discriminator-Id	{ 2 9 3 2 7 1 }
A-Event-Time	{ 2 9 3 2 7 13 }
A-Event-Type	{ 2 9 3 2 7 14 }
A-Incoming-Connection-Reject-Error-Counter	{ 2 9 3 2 7 73 }
A-Incoming-Connection-Reject-Error-Threshold	{ 2 9 3 2 7 90 }
A-Incoming-Connection-Requests-Counter	{ 2 9 3 2 7 74 }
A-Incoming-Connection-Requests-Threshold	{ 2 9 3 2 7 91 }
A-Incoming-Disconnect-Counter	{ 2 9 3 2 7 75 }
A-Incoming-Disconnect-Error-Counter	{ 2 9 3 2 7 76 }
A-Incoming-Disconnect-Error-Threshold	{ 2 9 3 2 7 92 }
A-Incoming-Protocol-Error-Counter	{ 2 9 3 2 7 77 }
A-Incoming-Protocol-Error-Threshold	{ 2 9 3 2 7 93 }
A-Intervals-Of-Day	{ 2 9 3 2 7 57 }
A-Log-Full-Action	{ 2 9 3 2 7 58 }
A-Log-Id	{ 2 9 3 2 7 2 }
A-Log-Record-Id	{ 2 9 3 2 7 3 }
A-Logging-Time	{ 2 9 3 2 7 59 }
A-Managed-Object-Class	{ 2 9 3 2 7 60 }
A-Managed-Object-Instance	{ 2 9 3 2 7 61 }
A-Max-Log-Size	{ 2 9 3 2 7 62 }
A-Member	{ 2 9 3 2 7 42 }
A-Monitored-Attributes	{ 2 9 3 2 7 15 }
A-Name-Binding	{ 2 9 3 2 7 63 }
A-Notification-Identifier	{ 2 9 3 2 7 16 }
A-Number-Of-Records	{ 2 9 3 2 7 64 }
A-Object-Class	{ 2 9 3 2 7 65 }
A-Octets-Received-Counter	{ 2 9 3 2 7 78 }
A-Octets-Received-Threshold	{ 2 9 3 2 7 94 }
A-Octets-Retransmitted-Error-Counter	{ 2 9 3 2 7 79 }
A-Octets-Retransmitted-Threshold	{ 2 9 3 2 7 95 }

Table 112 (Page 3 of 4). Object Identifiers for DMI Attributes

Attribute	Object Identifier
A-Octets-Sent-Counter	{ 2 9 3 2 7 80 }
A-Octets-Sent-Threshold	{ 2 9 3 2 7 96 }
A-Operational-State	{ 2 9 3 2 7 35 }
A-Outgoing-Connection-Reject-Error-Counter	{ 2 9 3 2 7 81 }
A-Outgoing-Connection-Reject-Error-Threshold	{ 2 9 3 2 7 97 }
A-Outgoing-Connection-Requests-Counter	{ 2 9 3 2 7 82 }
A-Outgoing-Connection-Requests-Threshold	{ 2 9 3 2 7 98 }
A-Outgoing-Disconnect-Counter	{ 2 9 3 2 7 83 }
A-Outgoing-Disconnect-Error-Counter	{ 2 9 3 2 7 84 }
A-Outgoing-Disconnect-Error-Threshold	{ 2 9 3 2 7 99 }
A-Outgoing-Protocol-Error-Counter	{ 2 9 3 2 7 85 }
A-Outgoing-Protocol-Error-Threshold	{ 2 9 3 2 7 100 }
A-Owner	{ 2 9 3 2 7 43 }
A-Packages	{ 2 9 3 2 7 66 }
A-Pdus-Received-Counter	{ 2 9 3 2 7 86 }
A-Pdus-Received-Threshold	{ 2 9 3 2 7 101 }
A-Pdus-Retransmitted-Error-Counter	{ 2 9 3 2 7 87 }
A-Pdus-Retransmitted-Error-Threshold	{ 2 9 3 2 7 102 }
A-Pdus-Sent-Counter	{ 2 9 3 2 7 88 }
A-Pdus-Sent-Threshold	{ 2 9 3 2 7 103 }
A-Peer	{ 2 9 3 2 7 44 }
A-Perceived-Severity	{ 2 9 3 2 7 17 }
A-Primary	{ 2 9 3 2 7 45 }
A-Probable-Cause	{ 2 9 3 2 7 18 }
A-Procedural-Status	{ 2 9 3 2 7 36 }
A-Proposed-Repair-Actions	{ 2 9 3 2 7 19 }
A-Provider-Object	{ 2 9 3 2 7 46 }
A-Relationship-Change-Definition	{ 2 9 3 2 7 20 }
A-Scheduler-Name	{ 2 9 3 2 7 67 }
A-Secondary	{ 2 9 3 2 7 47 }
A-Security-Alarm-Cause	{ 2 9 3 2 7 21 }
A-Security-Alarm-Detector	{ 2 9 3 2 7 22 }
A-Security-Alarm-Severity	{ 2 9 3 2 7 23 }
A-Service-Provider	{ 2 9 3 2 7 24 }
A-Service-User	{ 2 9 3 2 7 25 }
A-Source-Indicator	{ 2 9 3 2 7 26 }
A-Specific-Problems	{ 2 9 3 2 7 27 }
A-Standby-Status	{ 2 9 3 2 7 37 }

Table 112 (Page 4 of 4). Object Identifiers for DMI Attributes

Attribute	Object Identifier
A-Start-Time	{ 2 9 3 2 7 68 }
A-State-Change-Definition	{ 2 9 3 2 7 28 }
A-Stop-Time	{ 2 9 3 2 7 69 }
A-Supported-Features	{ 2 9 3 2 7 70 }
A-System-Id	{ 2 9 3 2 7 4 }
A-System-Title	{ 2 9 3 2 7 5 }
A-Threshold-Info	{ 2 9 3 2 7 29 }
A-Trend-Indication	{ 2 9 3 2 7 30 }
A-Unknown-Status	{ 2 9 3 2 7 38 }
A-Usage-State	{ 2 9 3 2 7 39 }
A-User-Object	{ 2 9 3 2 7 48 }
A-Week-Mask	{ 2 9 3 2 7 71 }

Object Identifiers for DMI Attribute Groups

Table 113. Object Identifiers for DMI Attribute Groups

Attribute Group	Object Identifier
A-Relationships	{ 2 9 3 2 8 2 }
A-State	{ 2 9 3 2 8 1 }

Object Identifiers for DMI Notifications

Table 114. Object Identifiers for DMI Notifications

Notification	Object Identifier
N-Attribute-Value-Change	{ 2 9 3 2 10 1 }
N-Communications-Alarm	{ 2 9 3 2 10 2 }
N-Environmental-Alarm	{ 2 9 3 2 10 3 }
N-Equipment-Alarm	{ 2 9 3 2 10 4 }
N-Integrity-Violation	{ 2 9 3 2 10 5 }
N-Object-Creation	{ 2 9 3 2 10 6 }
N-Object-Deletion	{ 2 9 3 2 10 7 }
N-Operational-Violation	{ 2 9 3 2 10 8 }
N-Physical-Violation	{ 2 9 3 2 10 9 }
N-Processing-Error-Alarm	{ 2 9 3 2 10 10 }
N-Qualityof-Service-Alarm	{ 2 9 3 2 10 11 }
N-Relationship-Change	{ 2 9 3 2 10 12 }
N-Security-Service-Or-Mechanism-Violation	{ 2 9 3 2 10 13 }
N-State-Change	{ 2 9 3 2 10 14 }
N-Time-Domain-Violation	{ 2 9 3 2 10 15 }

Object Identifiers for DMI Parameters

Table 115. Object Identifiers for DMI Parameters

Parameter	Object Identifier
S-Miscellaneous-Error	{ 2 9 3 2 5 1 }

Object Identifiers for DMI Name Bindings

Table 116. Object Identifiers for DMI Name Bindings

Name Binding	Object Identifier
B-Discriminator-System	{ 2 9 3 2 6 1 }
B-Log-System	{ 2 9 3 2 6 2 }
B-Log-Record-Log	{ 2 9 3 2 6 3 }

Object Identifiers for DMI Packages

Table 117 (Page 1 of 2). Object Identifiers for DMI Packages

Package	Object Identifier
P-Additional-Information-Package	{ 2 9 3 2 4 18 }
P-Additional-Text-Package	{ 2 9 3 2 4 19 }
P-Administrative-State-Package	{ 2 9 3 2 4 14 }
P-Allomorphic-Package	{ 2 9 3 2 4 17 }
P-Attribute-Identifier-List-Package	{ 2 9 3 2 4 20 }
P-Attribute-List-Package	{ 2 9 3 2 4 21 }
P-Availability-Status-Package	{ 2 9 3 2 4 22 }
P-Back-Up-Destination-List-Package	{ 2 9 3 2 4 9 }
P-Back-Up-Object-Package	{ 2 9 3 2 4 3 }
P-Backed-Up-Status-Package	{ 2 9 3 2 4 2 }
P-Correlated-Notifications-Package	{ 2 9 3 2 4 23 }
P-Daily-Scheduling	{ 2 9 3 2 4 25 }
P-Duration	{ 2 9 3 2 4 26 }
P-Event-Time-Package	{ 2 9 3 2 4 11 }
P-External-Scheduler	{ 2 9 3 2 4 27 }
P-Finite-Log-Size-Package	{ 2 9 3 2 4 12 }
P-Log-Alarm-Package	{ 2 9 3 2 4 13 }
P-Mode-Package	{ 2 9 3 2 4 10 }
P-Monitored-Attributes-Package	{ 2 9 3 2 4 7 }
P-Notification-Identifier-Package	{ 2 9 3 2 4 24 }
P-Packages-Package	{ 2 9 3 2 4 16 }
P-Proposed-Repair-Actions-Package	{ 2 9 3 2 4 8 }
P-Source-Indicator-Package	{ 2 9 3 2 4 28 }
P-Specific-Problems-Package	{ 2 9 3 2 4 1 }

Table 117 (Page 2 of 2). Object Identifiers for DMI Packages

Package	Object Identifier
P-State-Change-Definition-Package	{ 2 9 3 2 4 6 }
P-Supported-Features-Package	{ 2 9 3 2 4 15 }
P-Threshold-Info-Package	{ 2 9 3 2 4 5 }
P-Trend-Indication-Package	{ 2 9 3 2 4 4 }
P-Weekly-Scheduling	{ 2 9 3 2 4 29 }

Information Syntax Tables

This section contains DMI information syntax tables.

DMI Attribute Value Syntaxes

Table 118 (Page 1 of 3). DMI Attribute Value Syntaxes

Attribute Type	Attribute Syntax
A-Active-Destination	Object (Destination)
A-Additional-Information	Object (Additional-Information)
A-Additional-Text	String (Graphic-String)
A-Administrative-State	Enum (Administrative-State)
A-Alarm-Status	Object (Alarm-Status)
A-Allomorphs	Object (Allomorphs)
A-Attribute-Identifier-List	Object (Attribute-Identifier-List)
A-Attribute-List	Object (Attribute-List)
A-Attribute-Value-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)
A-Availability-Status	Object (Availability-Status)
A-Back-Up-Destination-List	Object (Back-Up-Destination-List)
A-Back-Up-Object	Object (Back-Up-Relationship-Object)
A-Backed-Up-Object	Object (Back-Up-Relationship-Object)
A-Backed-Up-Status	Boolean
A-Capacity-Alarm-Threshold	Object (Capacity-Alarm-Threshold)
A-Confirmed-Mode	Boolean
A-Control-Status	Object (Control-Status)
A-Correlated-Notifications	Object (Setof-Correlated-Notifications)
A-Counter	Integer
A-Counter--Threshold	Object (Setof-Counter-Threshold)
A-Current-Log-Size	Integer
A-Destination	Object (Destination)
A-Discriminator-Construct	Object (CMISFilter)
A-Discriminator-Id	Object (Simple-Name-Type)
A-Event-Time	String (Generalized-Time)
A-Event-Type	Object (Event-Type-Id)

Table 118 (Page 2 of 3). DMI Attribute Value Syntaxes

Attribute Type	Attribute Syntax
A-Gauge	Object (Observed-Value)
A-Gauge--Threshold	Object (Setof-Gauge-Threshold)
A-Intervals-Of-Day	Object (Setof-Intervals-Of-Day)
A-Log-Full-Action	Enum (Log-Full-Action)
A-Log-Id	Object (Simple-Name-Type)
A-Log-Record-Id	Object (Simple-Name-Type)
A-Logging-Time	String (Generalized-Time)
A-Managed-Object-Class	Object (Object-Class)
A-Managed-Object-Instance	Object (Object-Instance)
A-Max-Log-Size	Integer
A-Member	Object (Group-Objects)
A-Monitored-Attributes	Object (Monitored-Attributes)
A-Name-Binding	String (Object-Identifier)
A-Notification-Identifier	Integer
A-Number-Of-Records	Integer
A-Object-Class	Object (Object-Class)
A-Operational-State	Enum (Operational-State)
A-Owner	Object (Group-Objects)
A-Packages	Object (Packages)
A-Peer	Object (Back-Up-Relationship-Object)
A-Perceived-Severity	Enum (Perceived-Severity)
A-Primary	Object (Setof-Prioritised-Object)
A-Probable-Cause	Object (Probable-Cause)
A-Procedural-Status	Object (Procedural-Status)
A-Proposed-Repair-Actions	Object (Proposed-Repair-Actions)
A-Provider-Object	Object (Setof-Prioritised-Object)
A-Relationship-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)
A-Scheduler-Name	Object (Object-Instance)
A-Secondary	Object (Setof-Prioritised-Object)
A-Security-Alarm-Cause	String (Object-Identifier)
A-Security-Alarm-Detector	Object (Security-Alarm-Detector)
A-Security-Alarm-Severity	Enum (Perceived-Severity)
A-Service-Provider	Object (Service-User)
A-Service-User	Object (Service-User)
A-Source-Indicator	Enum (Source-Indicator)
A-Specific-Problems	Object (Specific-Problems)
A-Standby-Status	Integer
A-Start-Time	String (Generalized-Time)

Table 118 (Page 3 of 3). DMI Attribute Value Syntaxes

Attribute Type	Attribute Syntax
A-State-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)
A-Stop-Time	Object (Stop-Time)
A-Supported-Features	Object (Setof-Supported-Features)
A-System-Id	Object (System-Id)
A-System-Title	Object (System-Title)
A-Threshold-Info	Object (Threshold-Info)
A-Tide-Mark	Object (Tide-Mark-Info)
A-Trend-Indication	Enum (Trend-Indication)
A-Unknown-Status	Boolean
A-Usage-State	Enum (Usage-State)
A-User-Object	Object (Setof-Prioritised-Object)
A-Week-Mask	Object (Setof-Week-Mask)

DMI Notification Information Syntaxes

Table 119. DMI Notification Information Syntaxes

Notification Type	Information Syntax	Reply Syntax
N-Attribute-Value-Change	Object (Attribute-Value-Change-Info)	-
N-Communications-Alarm	Object (Alarm-Info)	-
N-Environmental-Alarm	Object (Alarm-Info)	-
N-Equipment-Alarm	Object (Alarm-Info)	-
N-Integrity-Violation	Object (Security-Alarm-Info)	-
N-Object-Creation	Object (Object-Info)	-
N-Object-Deletion	Object (Object-Info)	-
N-Operational-Violation	Object (Security-Alarm-Info)	-
N-Physical-Violation	Object (Security-Alarm-Info)	-
N-Processing-Error-Alarm	Object (Alarm-Info)	-
N-Qualityof-Service-Alarm	Object (Alarm-Info)	-
N-Relationship-Change	Object (Relationship-Change-Info)	-
N-Security-Service-Or-Mechanism-Violation	Object (Security-Alarm-Info)	-
N-State-Change	Object (State-Change-Info)	-
N-Time-Domain-Violation	Object (Security-Alarm-Info)	-

DMI Parameter Value Syntaxes

Table 120. DMI Parameter Value Syntaxes

Parameter Type	Parameter Syntax
S-Miscellaneous-Error	Null

OM Attribute Tables

This section contains tables for OM attributes.

Additional-Information

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 121. OM Attributes of an Additional-Information

OM Attribute	Value Syntax	Value Length	Value Number
management-Extension	Object (Management-Extension)	-	0 - more

Alarm-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 122. OM Attributes of an Alarm-Info

OM Attribute	Value Syntax	Value Length	Value Number
probable-Cause	Object (Probable-Cause)	-	1
specific-Problems	Object (Specific-Problems)	-	0 - 1
perceived-Severity	Enum (Perceived-Severity)	-	1
backed-Up-Status	Boolean	-	0 - 1
back-Up-Object	Object (Object-Instance)	-	0 - 1
trend-Indication	Enum (Trend-Indication)	-	0 - 1
threshold-Info	Object (Threshold-Info)	-	0 - 1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
state-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)	-	0 - 1
monitored-Attributes	Object (Monitored-Attributes)	-	0 - 1
proposed-Repair-Actions	Object (Proposed-Repair-Actions)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Alarm-Status

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 123. OM Attributes of an Alarm-Status

OM Attribute	Value Syntax	Value Length	Value Number
alarm-Status	Integer	-	0 - more

Allomorphs

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 124. OM Attributes of an Allomorphs

OM Attribute	Value Syntax	Value Length	Value Number
object-Class	Object (Object-Class)	-	0 - more

Attribute-Identifier-List

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 125. OM Attributes of an Attribute-Identifier-List

OM Attribute	Value Syntax	Value Length	Value Number
attribute-Id	Object (Attribute-Id)	-	0 - more

Attribute-List

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 126. OM Attributes of an Attribute-List

OM Attribute	Value Syntax	Value Length	Value Number
attribute	Object (Attribute)	-	0 - more

Setof-Attribute-Value-Change-Definition

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 127. OM Attributes of a Setof-Attribute-Value-Change-Definition

OM Attribute	Value Syntax	Value Length	Value Number
attribute-Value-Change-Definition	Object (Attribute-Value-Change-Definition)	-	0 - more

Attribute-Value-Change-Definition

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 128. OM Attributes of an Attribute-Value-Change-Definition

OM Attribute	Value Syntax	Value Length	Value Number
attribute-ID	Object (Attribute-Id)	-	1
old-Attribute-Value	Any	-	0 - 1
new-Attribute-Value	Any	-	1

Attribute-Value-Change-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 129. OM Attributes of an Attribute-Value-Change-Info

OM Attribute	Value Syntax	Value Length	Value Number
source-Indicator	Enum (Source-Indicator)	-	0 - 1
attribute-Identifier-List	Object (Attribute-Identifier-List)	-	0 - 1
attribute-Value-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)	-	1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Availability-Status

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 130. OM Attributes of an Availability-Status

OM Attribute	Value Syntax	Value Length	Value Number
availability-Status	Integer	-	0 - more

Back-Up-Destination-List

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 131. OM Attributes of a Back-Up-Destination-List

OM Attribute	Value Syntax	Value Length	Value Number
ae-Title	Object (AE-Title)	-	0 - more

Back-Up-Relationship-Object

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 132. OM Attributes of a Back-Up-Relationship-Object

OM Attribute	Value Syntax	Value Length	Value Number
object-Name	Object (Object-Instance)	-	0 - 1
no-Object	Null	-	0 - 1

Capacity-Alarm-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 133. OM Attributes of a Capacity-Alarm-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
capacity-Alarm-Threshold	Integer	-	0 - more

Control-Status

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 134. OM Attributes of a Control-Status

OM Attribute	Value Syntax	Value Length	Value Number
control-Status	Integer	-	0 - more

Setof-Correlated-Notifications

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 135. OM Attributes of a Setof-Correlated-Notifications

OM Attribute	Value Syntax	Value Length	Value Number
correlated-Notifications	Object (Correlated-Notifications)	-	0 - more

Correlated-Notifications

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 136. OM Attributes of a Correlated-Notifications

OM Attribute	Value Syntax	Value Length	Value Number
correlated-Notifications	Object (Correlated-Notifications-1)	-	1
source-Object-Inst	Object (Object-Instance)	-	0 - 1

Correlated-Notifications-1

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 137. OM Attributes of a Correlated-Notifications-1

OM Attribute	Value Syntax	Value Length	Value Number
notification-Identifier	Integer	-	0 - more

Setof-Counter-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 138. OM Attributes of a Setof-Counter-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
counter-Threshold	Object (Counter-Threshold)	-	0 - more

Counter-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 139. OM Attributes of a Counter-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
comparison-Level	Integer	-	1
offset-Value	Integer	-	1
notification-On-Off	Boolean	-	1

Destination

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 140. OM Attributes of a Destination

OM Attribute	Value Syntax	Value Length	Value Number
single	Object (AE-Title)	-	0 - 1
multiple	Object (Multiple)	-	0 - 1

Multiple

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 141. OM Attributes of a Multiple

OM Attribute	Value Syntax	Value Length	Value Number
ae-Title	Object (AE-Title)	-	0 - more

Setof-Gauge-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 142. OM Attributes of a Setof-Gauge-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
gauge-Threshold	Object (Gauge-Threshold)	-	0 - more

Gauge-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 143. OM Attributes of a Gauge-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
notify-Low	Object (Notify-Threshold)	-	1
notify-High	Object (Notify-Threshold)	-	1

Group-Objects

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 144. OM Attributes of a Group-Objects

OM Attribute	Value Syntax	Value Length	Value Number
object-Instance	Object (Object-Instance)	-	0 - more

Setof-Intervals-Of-Day

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 145. OM Attributes of a Setof-Intervals-Of-Day

OM Attribute	Value Syntax	Value Length	Value Number
intervals-Of-Day	Object (Intervals-Of-Day)	-	0 - more

Intervals-Of-Day

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 146. OM Attributes of a Intervals-Of-Day

OM Attribute	Value Syntax	Value Length	Value Number
interval-Start	Object (Time24)	-	1
interval-End	Object (Time24)	-	1

Management-Extension

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 147. OM Attributes of a Management-Extension

OM Attribute	Value Syntax	Value Length	Value Number
identifier	String (Object-Identifier)	-	1
significance	Boolean	-	1
information	Any	-	1

Monitored-Attributes

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 148. OM Attributes of a Monitored-Attributes

OM Attribute	Value Syntax	Value Length	Value Number
attribute	Object (Attribute)	-	0 - more

Notify-Threshold

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 149. OM Attributes of a Notify-Threshold

OM Attribute	Value Syntax	Value Length	Value Number
threshold	Object (Observed-Value)	-	1
notify-On-Off	Boolean	-	1

Object-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 150. OM Attributes of a Object-Info

OM Attribute	Value Syntax	Value Length	Value Number
source-Indicator	Enum (Source-Indicator)	-	0 - 1
attribute-List	Object (Attribute-List)	-	0 - 1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Observed-Value

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 151. OM Attributes of an Observed-Value

OM Attribute	Value Syntax	Value Length	Value Number
integer	Integer	-	0 - 1
real	Real	-	0 - 1

Packages

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 152. OM Attributes of a Packages

OM Attribute	Value Syntax	Value Length	Value Number
packages	String (Object-Identifier)	-	0 - more

Setof-Prioritised-Object

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 153. OM Attributes of a Setof-Prioritised-Object

OM Attribute	Value Syntax	Value Length	Value Number
prioritised-Object	Object (Prioritised-Object)	-	0 - more

Prioritised-Object

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 154. OM Attributes of a Prioritised-Object

OM Attribute	Value Syntax	Value Length	Value Number
object	Object (Object-Instance)	-	1
priority	Integer	-	1

Probable-Cause

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 155. OM Attributes of a Probable-Cause

OM Attribute	Value Syntax	Value Length	Value Number
global-Value	String (Object-Identifier)	-	0 - 1
local-Value	Integer	-	0 - 1

Procedural-Status

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 156. OM Attributes of a Procedural-Status

OM Attribute	Value Syntax	Value Length	Value Number
procedural-Status	Integer	-	0 - more

Proposed-Repair-Actions

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 157. OM Attributes of a Proposed-Repair-Actions

OM Attribute	Value Syntax	Value Length	Value Number
specific-Identifier	Object (Specific-Identifier)	-	0 - more

Relationship-Change-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 158. OM Attributes of a Relationship-Change-Info

OM Attribute	Value Syntax	Value Length	Value Number
source-Indicator	Enum (Source-Indicator)	-	0 - 1
attribute-Identifier-List	Object (Attribute-Identifier-List)	-	0 - 1
relationship-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)	-	1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Security-Alarm-Detector

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 159. OM Attributes of a Security-Alarm-Detector

OM Attribute	Value Syntax	Value Length	Value Number
mechanism	String (Object-Identifier)	-	0 - 1
object	Object (Object-Instance)	-	0 - 1
application	Object (AE-Title)	-	0 - 1

Security-Alarm-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 160. OM Attributes of a Security-Alarm-Info

OM Attribute	Value Syntax	Value Length	Value Number
security-Alarm-Cause	String (Object-Identifier)	-	1
security-Alarm-Severity	Enum (Perceived-Severity)	-	1
security-Alarm-Detector	Object (Security-Alarm-Detector)	-	1
service-User	Object (Service-User)	-	1
service-Provider	Object (Service-User)	-	1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Service-User

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 161. OM Attributes of a Service-User

OM Attribute	Value Syntax	Value Length	Value Number
identifier	String (Object-Identifier)	-	1
details	Any	-	1

Simple-Name-Type

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 162. OM Attributes of a Simple-Name-Type

OM Attribute	Value Syntax	Value Length	Value Number
number	Integer	-	0 - 1
string	String (Graphic-String)	-	0 - 1

Specific-Identifier

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 163. OM Attributes of a Specific-Identifier

OM Attribute	Value Syntax	Value Length	Value Number
specific-Identifier-OBJECT-IDENTIFIER-1	String (Object-Identifier)	-	0 - 1
specific-Identifier-INTEGER-2	Integer	-	0 - 1

Specific-Problems

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 164. OM Attributes of a Specific-Problems

OM Attribute	Value Syntax	Value Length	Value Number
specific-Identifier	Object (Specific-Identifier)	-	0 - more

State-Change-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 165. OM Attributes of a State-Change-Info

OM Attribute	Value Syntax	Value Length	Value Number
source-Indicator	Enum (Source-Indicator)	-	0 - 1
attribute-Identifier-List	Object (Attribute-Identifier-List)	-	0 - 1
state-Change-Definition	Object (Setof-Attribute-Value-Change-Definition)	-	1
notification-Identifier	Integer	-	0 - 1
correlated-Notifications	Object (Setof-Correlated-Notifications)	-	0 - 1
additional-Text	String (Graphic-String)	-	0 - 1
additional-Information	Object (Additional-Information)	-	0 - 1

Stop-Time

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 166. OM Attributes of a Stop-Time

OM Attribute	Value Syntax	Value Length	Value Number
specific	String (Generalized-Time)	-	0 - 1
continual	Null	-	0 - 1

Setof-Supported-Features

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 167. OM Attributes of a Setof-Supported-Features

OM Attribute	Value Syntax	Value Length	Value Number
supported-Features	Object (Supported-Features)	-	0 - more

Supported-Features

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 168. OM Attributes of a Supported-Features

OM Attribute	Value Syntax	Value Length	Value Number
feature-Identifier	String (Object-Identifier)	-	1
feature-Info	Any	-	1

System-Id

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 169. OM Attributes of a System-Id

OM Attribute	Value Syntax	Value Length	Value Number
name	String (Graphic-String)	-	0 - 1
number	Integer	-	0 - 1
nothing	Null	-	0 - 1

System-Title

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 170. OM Attributes of a System-Title

OM Attribute	Value Syntax	Value Length	Value Number
distinguished-Name	Object (Distinguished-Name)	-	0 - 1
oid	String (Object-Identifier)	-	0 - 1
nothing	Null	-	0 - 1

Threshold-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 171. OM Attributes of a Threshold-Info

OM Attribute	Value Syntax	Value Length	Value Number
triggered-Threshold	Object (Attribute-Id)	-	1
observed-Value	Object (Observed-Value)	-	1
threshold-Level	Object (Threshold-Level-Ind)	-	0 - 1
arm-Time	String (Generalized-Time)	-	0 - 1

Threshold-Level-Ind

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 172. OM Attributes of a Threshold-Level-Ind

OM Attribute	Value Syntax	Value Length	Value Number
up	Object (Up)	-	0 - 1
down	Object (Down)	-	0 - 1

Up

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 173. OM Attributes of a Up

OM Attribute	Value Syntax	Value Length	Value Number
high	Object (Observed-Value)	-	1
low	Object (Observed-Value)	-	0 - 1

Down

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 174. OM Attributes of a Down

OM Attribute	Value Syntax	Value Length	Value Number
high	Object (Observed-Value)	-	1
low	Object (Observed-Value)	-	1

Tide-Mark

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Exactly one OM attribute is permitted in an instance of this OM class.

Table 175. OM Attributes of a Tide-Mark

OM Attribute	Value Syntax	Value Length	Value Number
max-Tide-Mark	Object (Observed-Value)	-	0 - 1
min-Tide-Mark	Object (Observed-Value)	-	0 - 1

Tide-Mark-Info

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 176 (Page 1 of 2). OM Attributes of a Tide-Mark-Info

OM Attribute	Value Syntax	Value Length	Value Number
current-Tide-Mark	Object (Tide-Mark)	-	1
previous-Tide-Mark	Object (Tide-Mark)	-	1

Table 176 (Page 2 of 2). OM Attributes of a Tide-Mark-Info

OM Attribute	Value Syntax	Value Length	Value Number
reset-Time	String (Generalized-Time)	-	1

Time24

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 177. OM Attributes of a Time24

OM Attribute	Value Syntax	Value Length	Value Number
hour	Integer	-	1
minute	Integer	-	1

Setof-Week-Mask

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 178. OM Attributes of a Setof-Week-Mask

OM Attribute	Value Syntax	Value Length	Value Number
week-Mask	Object (Week-Mask)	-	0 - more

Week-Mask

An instance of this OM class has the OM attributes of its superclass - Object - and additionally the OM attributes listed below.

Table 179. OM Attributes of a Week-Mask

OM Attribute	Value Syntax	Value Length	Value Number
days-Of-Week	String (Bit-String)	7	1
intervals-Of-Day	Object (Setof-Intervals-Of-Day)	-	1

Value Lists

Administrative-State

Value List for Administrative-State is one of the following:

locked (0)
unlocked (1)
shutting-Down (2)

Alarm-Status

Value List for Alarm-Status is one of the following:

under-Repair (0)
critical (1)
major (2)
minor (3)
alarm-Outstanding (4)

Availability-Status

Value List for Availability-Status is one of the following:

in-Test (0)
failed (1)
power-Off (2)
off-Line (3)
off-Duty (4)
dependency (5)
degraded (6)
not-Installed (7)
log-Full (8)

Control-Status

Value List for Control-Status is one of the following:

subject-To-Test (0)
part-Of-Services-Locked (1)
reserved-For-Test (2)
suspended (3)

Log-Full-Action

Value List for Log-Full-Action. is one of the following:

wrap (0)
halt (1)

Max-Log-Size

Value List for Max-Log-Size is one of the following:

unlimited (0)

Operational-State

Value List for Operational-State is one of the following:

disabled (0)
enabled (1)

Perceived-Severity

Value List for Perceived-Severity is one of the following:

indeterminate (0)
critical (1)
major (2)
minor (3)
warning (4)
cleared (5)

Priority

Value List for Priority is one of the following:

lowest (0)
highest (127)

Probable-Cause

Value List for Probable-Cause is one of the following:

adapter-Error ({ 2 9 3 2 0 0 1 })
application-Subsystem-Failure ({ 2 9 3 2 0 0 2 })
bandwidth-Reduced ({ 2 9 3 2 0 0 3 })
call-Establishment-Error ({ 2 9 3 2 0 0 4 })
communications-Protocol-Error ({ 2 9 3 2 0 0 5 })
communications-Subsystem-Failure ({ 2 9 3 2 0 0 6 })
configuration-Or-Customization-Error ({ 2 9 3 2 0 0 7 })
congestion ({ 2 9 3 2 0 0 8 })
corrupt-Data ({ 2 9 3 2 0 0 9 })
cpu-Cycles-Limit-Exceeded ({ 2 9 3 2 0 0 10 })
d-TE-DCE_Interface-Error ({ 2 9 3 2 0 0 13 })
data-Set-Or-Modem-Error ({ 2 9 3 2 0 0 11 })
degraded-Signal ({ 2 9 3 2 0 0 12 })
enclosure-Door-Open ({ 2 9 3 2 0 0 14 })
equipment-Malfunction ({ 2 9 3 2 0 0 15 })
excessive-Vibration ({ 2 9 3 2 0 0 16 })
file-Error ({ 2 9 3 2 0 0 17 })
fire-Detected ({ 2 9 3 2 0 0 18 })
flood-Detected ({ 2 9 3 2 0 0 19 })
framing-Error ({ 2 9 3 2 0 0 20 })
heating-Or-Ventilation-Or-Cooling-System-Problem ({ 2 9 3 2 0 0 21 })
humidity-Unacceptable ({ 2 9 3 2 0 0 22 })
input-Device-Error ({ 2 9 3 2 0 0 24 })
input-Output-Device-Error ({ 2 9 3 2 0 0 23 })
l-AN_Error ({ 2 9 3 2 0 0 25 })
leak-Detected ({ 2 9 3 2 0 0 26 })
local-Node-Transmission-Error ({ 2 9 3 2 0 0 27 })
loss-Of-Frame ({ 2 9 3 2 0 0 28 })
loss-Of-Signal ({ 2 9 3 2 0 0 29 })
material-Supply-Exhausted ({ 2 9 3 2 0 0 30 })
multiplexer-Problem ({ 2 9 3 2 0 0 31 })
output-Device-Error ({ 2 9 3 2 0 0 33 })
out-Of-Memory ({ 2 9 3 2 0 0 32 })

performance-Degraded ({ 2 9 3 2 0 0 34 })
power-Problem ({ 2 9 3 2 0 0 35 })
pressure-Unacceptable ({ 2 9 3 2 0 0 36 })
processor-Problem ({ 2 9 3 2 0 0 37 })
pump-Failure ({ 2 9 3 2 0 0 38 })
queue-Size-Exceeded ({ 2 9 3 2 0 0 39 })
receive-Failure ({ 2 9 3 2 0 0 40 })
receiver-Failure ({ 2 9 3 2 0 0 41 })
remote-Node-Transmission-Error ({ 2 9 3 2 0 0 42 })
resource-At-Or-Nearing-Capacity ({ 2 9 3 2 0 0 43 })
response-Time-Excessive ({ 2 9 3 2 0 0 44 })
retransmission-Rate-Excessive ({ 2 9 3 2 0 0 45 })
software-Error ({ 2 9 3 2 0 0 46 })
software-Program-Abnormally-Terminated ({ 2 9 3 2 0 0 47 })
software-Program-Error ({ 2 9 3 2 0 0 48 })
storage-Capacity-Problem ({ 2 9 3 2 0 0 49 })
temperature-Unacceptable ({ 2 9 3 2 0 0 50 })
threshold-Crossed ({ 2 9 3 2 0 0 51 })
timing-Problem ({ 2 9 3 2 0 0 52 })
toxic-Leak-Detected ({ 2 9 3 2 0 0 53 })
transmit-Failure ({ 2 9 3 2 0 0 54 })
transmitter-Failure ({ 2 9 3 2 0 0 55 })
underlying-Resource-Unavailable ({ 2 9 3 2 0 0 56 })
version-Mismatch ({ 2 9 3 2 0 0 57 })

Procedural-Status

Value List for Procedural-Status is one of the following:

initialization-Required (0)
not-Initialized (1)
initializing (2)
reporting (3)
terminating (4)

Security-Alarm-Cause

Value List for Security-Alarm-Cause is one of the following:

authentication-Failure ({ 2 9 3 2 0 1 1 })
breach-Of-Confidentiality ({ 2 9 3 2 0 1 2 })
cable-Tamper ({ 2 9 3 2 0 1 3 })
delayed-Information ({ 2 9 3 2 0 1 4 })
denial-Of-Service ({ 2 9 3 2 0 1 5 })
duplicate-Information ({ 2 9 3 2 0 1 6 })
information-Missing ({ 2 9 3 2 0 1 7 })
information-Modification-Detected ({ 2 9 3 2 0 1 8 })
information-Out-Of-Sequence ({ 2 9 3 2 0 1 9 })
intrusion-Detection ({ 2 9 3 2 0 1 10 })
key-Expired ({ 2 9 3 2 0 1 11 })
non-Repudiation-Failure ({ 2 9 3 2 0 1 12 })
out-Of-Hours-Activity ({ 2 9 3 2 0 1 13 })
out-Of-Service ({ 2 9 3 2 0 1 14 })
procedural-Error ({ 2 9 3 2 0 1 15 })
unauthorized-Access-Attempt ({ 2 9 3 2 0 1 16 })

unexpected-Information ({ 2 9 3 2 0 1 17 })
unspecified-Reason ({ 2 9 3 2 0 1 18 })

Source-Indicator

Value List for Source-Indicator is one of the following:

resource-Operation (0)
management-Operation (1)
unknown (2)

Standby-Status

Value List for Standby-Status is one of the following:

hot-Standby (0)
cold-Standby (1)
providing-Service (2)

Trend-Indication

Value List for Trend-Indication is one of the following:

less-Severe (0)
no-Change (1)
more-Severe (2)

Usage-State

Value List for Usage-State is one of the following:

idle (0)
active (1)
busy (2)

Days-Of-Week

Value List for Days-Of-Week is one of the following:

sunday (0)
monday (1)
tuesday (2)
wednesday (3)
thursday (4)
friday (5)
saturday (6)

Chapter 6. Using NetView for AIX GTM Data Structures

The data strutures used by the GTM API are defined in the nvotTypes.h file.

The **don't care** values defined in the API are:

- NULL: for pointers
- -1: for integers

The string fields that are stored in the NetView for AIX object database will be truncated to 256 characters. String fields in the GTM database can have unlimited length.

Basic Structures

Structures are used to represent the GTM entities. The structures contain components of three types:

- table structures, which define relevant variables for each entity type
- type structures, which define low-level variables for use in basic structures and table structures
- standard C-language data types such as **int** and **char**

For each entity, a structure is defined to represent that entity, and a second structure is defined to represent a list.

Vertex

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotVertexAttrType vertexAttr;
} nvotVertex;

typedef struct {
  int count;
  nvotVertex *vertex;
} nvotVertexList;
```

Graph

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotGraphAttrType graphAttr;
} nvotGraph;

typedef struct {
  int count;
  nvotGraph *graph;
} nvotGraphList;
```

Box

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotGraphAttrType boxAttr;
} nvotBox;
```

```
typedef struct {
  int count;
  nvotBox *box;
} nvotBoxList;
```

Arc

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotArcAttrType arcAttr;
} nvotArc;
```

```
typedef struct {
  int count;
  nvotArc *arc;
} nvotArcList;
```

Sap

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotSapAttrType sapAttr;
} nvotSap;
```

```
typedef struct {
  int count;
  nvotSap *sap;
} nvotSapList;
```

Simple Connection

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotScAttrType scAttr;
} nvotSimpleConnection;
```

```
typedef struct {
  int count;
  nvotSimpleConnection *simpleConnection;
} nvotSimpleConnectionList;
```

Underlying Connection

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotUlcAttrType ulcAttr;
} nvotUnderlyingConnection;
```

```
typedef struct {
  int count;
  nvotUnderlyingConnection *underlyingConnection;
} nvotUnderlyingConnectionList;
```

Underlying Arc

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotUlaAttrType ulaAttr;
} nvotUnderlyingArc;

typedef struct {
  int count;
  nvotUnderlyinfArc *underlyingArc;
} nvotUnderlyingArcList;
```

Members

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotMembersAttrType membersAttr;
} nvotMembers;

typedef struct {
  int count;
  nvotMembers *member;
} nvotMembersList;
```

Member Arcs

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotMaAttrType maAttr;
} nvotMemberArcs;

typedef struct {
  int count;
  nvotMemberArc *memberArc;
} nvotMemberArcsList;
```

Attached Arcs

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotAaAttrType aaAttr;
} nvotAttachedArcs;

typedef struct {
  int count;
  nvotAttachedArcs *attachedArc;
} nvotAttachedArcsList;
```

Additional Members

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotNameBindingType nameBinding;
  nvotAddMembersAttrType addMembersAttr;
} nvotAdditionalMembers;

typedef struct {
  int count;
  nvotAdditionalMembers *additionalMember;
} nvotAdditionalMembersList;
```

Additional Graph

```
typedef struct {
  nvotOperationType operation;
  nvotAttributeBitmapType validAttributes;
  nvotAddGraphAttrType addGraphAttr;
} nvotAdditionalGraph;

typedef struct {
  int count;
  nvotAddGraph *additionalGraph;
} nvotAdditionalGraphList;
```

Table Structures

A table structure is defined for use in each basic structure. There is no box table structure; the box basic structure uses the graph table structure.

nvotVertexAttrType

```
typedef struct {
  nvotVertexProtocolType        vertexProtocol;
  char *                        vertexName;
  nvotOwnerType                 vertexMine;
  char *                        vertexLocation;
  nvotOctetString               vertexManagementExtension;
  nvotOctetString               vertexManagementAddr;
  nvotOperationalStateType      vertexOperationalState;
  nvotUnknownStatusType         vertexUnknownStatus;
  nvotAvailabilityStatusType    vertexAvailabilityStatus;
  nvotAlarmStatusType           vertexAlarmStatus;
  char *                        vertexLabel;
  char *                        vertexIcon;
} nvotVertexAttrType;
```

nvotGraphAttrType

```
typedef struct {
  nvotGraphType                graphType;
  nvotGraphProtocolType        graphProtocol;
  char *                       graphName;
  nvotLayoutType               layoutAlgorithm;
  char *                       userDefinedLayout;
  char *                       graphLocation;
  char *                       backgroundMap;
  nvotOctetString              graphManagementExtension;
  nvotOctetString              graphManagementAddr;
  nvotBooleanType              isRoot;
  char *                       graphLabel;
  char *                       graphIcon;
} nvotGraphAttrType;
```

nvotArcAttrType

```
typedef struct {
  nvotProtocolType              aEnpointProtocol;
  char *                        aEndpointName;
  nvotProtocolType              zEndpointProtocol;
  char *                        zEndpointName;
  int                           arcIndexId;
  int                           aDetailsIndexId;
  int                           zDetailsIndexId;
  nvotOctetString               arcManagementExtension;
  nvotOctetString               arcManagementAddr;
  nvotOperationalStateType      arcOperationalState;
  nvotUnknownStatusType         arcUnknownStatus;
  nvotAvailabilityStatusType    arcAvailabilityStatus;
  nvotAlarmStatusType           arcAlarmStatus;
} nvotArcAttrType;
```

nvotSapAttrType

```
typedef struct {
  nvotProtocolType              sapVertexProtocol;
  char *                        sapVertexName;
  nvotServiceType               sapServiceType;
  nvotProtocolType              sapProtocol;
  char *                        sapAddress;
} nvotSapAttrType;
```

nvotScAttrType

```
typedef struct {
  nvotProtocolType              localEndpointProtocol;
  char *                        localEndpointName;
  int                           simpleConnIndexId;
  char *                        simpleConnName;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              connectionPartnerProtocol;
  char *                        connectionPartnerName;
  nvotOctetString               simpleConnManagementExtension;
  nvotOctetString               simpleConnManagementAddr;
  char *                        simpleConnIcon;
  nvotOperationalStateType      simpleConnOperationalState;
  nvotUnknownStatusType         simpleConnUnknownStatus;
  nvotAvailabilityStatusType    simpleConnAvailabilityStatus;
  nvotAlarmStatusType           simpleConnAlarmStatus;
} nvotScAttrType;
```

nvotUlcAttrType

```
typedef struct {
  nvotProtocolType              ulcEndpointProtocol;
  char *                        ulcEndpointName;
  int                           ulcEndpointId;
  nvotUnderlyingKindType        underlyingConnectionKind;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              uconnEndpointProtocol;
  char *                        uconnEndpointName;
  int                           uconnSimpleConnId;
  nvotNameBindingType           nextSerialNameBinding;
  nvotProtocolType              nextSerialEndpointProtocol;
  char *                        nextSerialEndpointName;
  int                           nextSerialSimpleConnId;
} nvotUlcAttrType;
```

nvotUlaAttrType

```
typedef struct {
  nvotProtocolType              ulaAendpointProtocol;
  char *                        ulaAendpointName;
  nvotProtocolType              ulaZendpointProtocol;
  char *                        ulaZendpointName;
  int                           ulaArcIndexId;
  nvotUnderlyingKindType        underlyingArcKind;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              uconnAendpointProtocol;
  char *                        uconnAendpointName;
  nvotProtocolType              uconnZendpointProtocol;
  char *                        uconnZendpointName;
  int                           uconnArcIndexId;
  nvotNameBindingType           nextSerialNameBinding;
  nvotProtocolType              nextSerialAendpointProtocol;
  char *                        nextSerialAendpointName;
  nvotProtocolType              nextSerialZendpointProtocol;
  char *                        nextSerialZendpointName;
  int                           nextSerialArcIndexId;
} nvotUlaAttrType;
```

nvotMembersAttrType

```
typedef struct {
  nvotProtocolType              memberProtocol;
  char *                        memberName;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              memberComponentProtocol;
  char *                        memberComponentName;
  char *                        memberLabel;
  char *                        memberIcon;
} nvotMembersAttrType;
```

nvotMaAttrType

```
typedef struct {
  nvotProtocolType              maGraphProtocol;
  char *                        maGraphName;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              maAendpointProtocol;
  char *                        maAendpointName;
  nvotProtocolType              maZendpointProtocol;
  char *                        maZendpointName;
  int                           maArcIndexId;
  char *                        maLabel;
  char *                        maIcon;
} nvotMaAttrType;
```

nvotAaAttrType

```
typedef struct {
  nvotProtocolType              aaGraphProtocol;
  char *                        aaGraphName;
  nvotNameBindingType           nameBinding;
  nvotProtocolType              aaAendpointProtocol;
  char *                        aaAendpointName;
  nvotProtocolType              aaZendpointProtocol;
  char *                        aaZendpointName;
  int                           aaArcIndexId;
} nvotAaAttrType;
```

nvotAddMembersAttrType

```
typedef struct {
  nvotProtocolType              aMemberProtocol;
  char *                        aMemberName;
  int                           aMemberIndexId;
  int                           xCoordinate;
  int                           yCoordinate;
  int                           xGrid;
  int                           yGrid;
} nvotAddMembersAttrType;
```

nvotAddGraphAttrType

```
typedef struct {
  nvotProtocolType              addGraphProtocol;
  char *                        addGraphName;
  int                           addGraphIndexId;
  nvotGraphProtocolType         graphRootProtocol;
  char *                        graphRootName;
  char *                        graphDesc1;
  int                           graphDescrX;
  int                           graphDescrY;
  char *                        rootGraphLabel;
  char *                        rootGraphIcon;
} nvotAddGraphAttrType;
```

Type Structures

These structures are the basic building blocks used to create basic structures and table structures.

nvotVertexProtocolType

A new application could need a vertex protocol that is not defined in this enumeration. The GTM API does not verify that a vertex protocol is one of those defined in this list. However, the GTM API verifies that the protocol is a nonnegative number.

```
typedef enum {
  NONEXISTENT         = 0,
  OTHER_PROTOCOL      = 1,
  REGULAR1822         = 2,
  HDH1822             = 3,
  DDN_X25             = 4,
  RFC877_X25          = 5,
  ETHERNET_CSMACD     = 6,
  ISO88023_CSMACD     = 7,
  ISO88024_TOKENBUS   = 8,
  ISO88025_TOKENRING  = 9,
  ISO88026_MAN        = 10,
  STARLAN             = 11,
  PROTEON_10MBIT      = 12,
  PROTEON_80MBIT      = 13,
  HYPERCHANNEL        = 14,
  FDDI                = 15,
  LAPB                = 16,
  SDLC                = 17,
  DSL                 = 18,
  EL                  = 19,
  BASIC_ISDN          = 20,
  PRIMARY_ISDN        = 21,
  PROP_P_P_SERIAL     = 22,
  PPP                 = 23,
  SW_LOOPBACK         = 24,
  EON                 = 25,
  ETHERNET_3MBIT      = 26,
  NSIP                = 27,
  SLIP                = 28,
  ULTRA               = 29,
```

```
  DS3                 = 30,
  SIP                 = 31,
  FRAME_RELAY         = 32,
  BASE_STATION_80211  = 50,
  REMOTE_STATION_80211= 51,
  APPN_END_NODE       = 52,
  APPN_NETWORK_NODE   = 53,
  SNA_SESSION         = 54,
  SNA_TG              = 55,
  IP                  = 56,
  LANFDDI             = 57,
  LANTR               = 58,
  LAN                 = 59,
  BNS                 = 60,
  DATA_LINK_SWITCH    = 65,
  HUB8250             = 70,
  HUB8250_MODULE      = 71,
  HUB8250_PORT        = 72,
  HUB8250_TRUNK       = 73,
  LMU6000             = 74
} nvotVertexProtocolType;
```

nvotGraphProtocolType

```
typedef char * nvotGraphProtocolType;
```

nvotProtocolType

```
typedef union {
  nvotVertexProtocolType vertexProtocol;
  nvotGraphProtocolType graphProtocol;
} nvotProtocolType;
```

nvotTableType

```
typedef enum {
  ALL_TABLE                   = 1, /* used in get operation */
  VERTEX_TABLE                = 2,
  SIMPLE_CONNECTION_TABLE     = 3,
  UNDERLYING_CONNECTION_TABLE = 4,
  ARC_TABLE                   = 5,
  UNDERLYING_ARC_TABLE        = 6,
  GRAPH_TABLE                 = 7,
  MEMBERS_TABLE               = 8,
  MEMBER_ARC_TABLE            = 9,
  ATTACHED_ARCS_TABLE         = 10,
  ADDITIONAL_GRAPH_TABLE      = 11,
  ADDITIONAL_MEMBERS_TABLE    = 12,
  SAP_TABLE                   = 13
} nvotTableType;
```

nvotAttributeBitmapType

This is a bitmap structure (4 hexadecimal digits) that indicates which attributes are filled in the table attributes variable.

```
typedef unsigned int nvotAttributeBitmapType;
```

Vertex

```
#define   VERTEXPROTOCOL_ATTR                SET_BIT( 0 )
#define   VERTEXNAME_ATTR                    SET_BIT( 1 )
#define   reserved for future use            SET_BIT( 2 )
#define   VERTEXLABEL_ATTR                   SET_BIT( 3 )
#define   VERTEXMINE_ATTR                    SET_BIT( 4 )
#define   VERTEXLOCATION_ATTR                SET_BIT( 5 )
#define   VERTEXMANAGEMENTEXTENSION_ATTR     SET_BIT( 6 )
#define   VERTEXMANAGEMENTADDR_ATTR          SET_BIT( 7 )
#define   VERTEXICON_ATTR                    SET_BIT( 8 )
#define   VERTEXOPERATIONALSTATE_ATTR        SET_BIT( 9 )
#define   VERTEXUNKNOWNSTATUS_ATTR           SET_BIT( 10 )
#define   VERTEXAVAILABILITYSTATUS_ATTR      SET_BIT( 11 )
#define   VERTEXALARMSTATUS_ATTR             SET_BIT( 12 )
```

Graph

```
#define   GRAPHPROTOCOL_ATTR                 SET_BIT( 0 )
#define   GRAPHNAME_ATTR                     SET_BIT( 1 )
#define   GRAPHTYPE_ATTR                     SET_BIT( 2 )
#define   LAYOUTALGORITHM_ATTR               SET_BIT( 3 )
#define   USERDEFINEDLAYOUT_ATTR             SET_BIT( 4 )
#define   GRAPHLOCATION_ATTR                 SET_BIT( 5 )
#define   BACKGROUNDMAP_ATTR                 SET_BIT( 6 )
#define   GRAPHMANAGEMENTEXTENSION_ATTR      SET_BIT( 7 )
#define   GRAPHMANAGEMENTADDR_ATTR           SET_BIT( 8 )
#define   GRAPHICON_ATTR                     SET_BIT( 9 )
#define   ISROOT_ATTR                        SET_BIT( 10 )
#define   GRAPHLABEL_ATTR                    SET_BIT( 11 )
```

Arc

```
#define   AENDPOINTPROTOCOL_ATTR             SET_BIT( 0 )
#define   ZENDPOINTPROTOCOL_ATTR             SET_BIT( 1 )
#define   AENDPOINTNAME_ATTR                 SET_BIT( 2 )
#define   ZENDPOINTNAME_ATTR                 SET_BIT( 3 )
#define   ARCINDEXID_ATTR                    SET_BIT( 4 )
#define   reserved for future use            SET_BIT( 5 )
#define   reserved for future use            SET_BIT( 6 )
#define   ADETAILSINDEXID_ATTR               SET_BIT( 7 )
#define   ZDETAILSINDEXID_ATTR               SET_BIT( 8 )
#define   ARCMANAGEMENTEXTENSION_ATTR        SET_BIT( 9 )
#define   ARCMANAGEMENTADDR_ATTR             SET_BIT( 10 )
#define   reserved for future use            SET_BIT( 11 )
#define   ARCOPERATIONALSTATE_ATTR           SET_BIT( 12 )
#define   ARCUNKNOWNSTATUS_ATTR              SET_BIT( 13 )
#define   ARCAVAILABILITYSTATUS_ATTR         SET_BIT( 14 )
#define   ARCALARMSTATUS_ATTR                SET_BIT( 15 )
```

Sap

```
#define   SAPVERTEXPROTOCOL_ATTR              SET_BIT( 0 )
#define   SAPVERTEXNAME_ATTR                  SET_BIT( 1 )
#define   reserved for future use             SET_BIT( 2 )
#define   SAPPROTOCOL_ATTR                    SET_BIT( 3 )
#define   SAPADDRESS_ATTR                     SET_BIT( 4 )
#define   SAPSERVICETYPE_ATTR                 SET_BIT( 5 )
#define   SAPTYPE_ATTR                        SET_BIT( 6 )
```

Simple Connection

```
#define   LOCALENDPOINTPROTOCOL_ATTR          SET_BIT( 0 )
#define   LOCALENDPOINTNAME_ATTR              SET_BIT( 1 )
#define   SIMPLECONNINDEXID_ATTR              SET_BIT( 2 )
#define   reserved for future use             SET_BIT( 3 )
#define   reserved for future use             SET_BIT( 4 )
#define   CONNECTIONNAMEBINDING_ATTR          SET_BIT( 5 )
#define   CONNECTIONPARTNERPROTOCOL_ATTR      SET_BIT( 6 )
#define   CONNECTIONPARTNERNAME_ATTR          SET_BIT( 7 )
#define   SIMPLECONNMANAGEMENTEXTENSION_ATTR  SET_BIT( 8 )
#define   SIMPLECONNMANAGEMENTADDR_ATTR       SET_BIT( 9 )
#define   reserved for future use             SET_BIT( 10 )
#define   SIMPLECONNOPERATIONALSTATE_ATTR     SET_BIT( 11 )
#define   SIMPLECONNUNKNOWNSTATUS_ATTR        SET_BIT( 12 )
#define   SIMPLECONNAVAILABILITYSTATUS_ATTR   SET_BIT( 13 )
#define   SIMPLECONNALARMSTATUS_ATTR          SET_BIT( 14 )
```

Underlying Connection

```
#define   ULCENDPOINTPROTOCOL_ATTR            SET_BIT( 0 )
#define   ULCENDPOINTNAME_ATTR                SET_BIT( 1 )
#define   ULCENDPOINTID_ATTR                  SET_BIT( 2 )
#define   reserved for future use             SET_BIT( 3 )
#define   reserved for future use             SET_BIT( 4 )
#define   UNDERLYINGCONNECTIONKIND_ATTR       SET_BIT( 5 )
#define   UCONNSIMPLECONNNAMEBINDING_ATTR     SET_BIT( 6 )
#define   UCONNENDPOINTPROTOCOL_ATTR          SET_BIT( 7 )
#define   UCONNENDPOINTNAME_ATTR              SET_BIT( 8 )
#define   UCONNSIMPLECONNID_ATTR              SET_BIT( 9 )
#define   reserved for future use             SET_BIT( 10 )
#define   NEXTSERIALNAMEBINDING_ATTR          SET_BIT( 11 )
#define   NEXTSERIALENDPOINTPROTOCOL_ATTR     SET_BIT( 12 )
#define   NEXTSERIALENDPOINTNAME_ATTR         SET_BIT( 13 )
#define   NEXTSERIALSIMPLECONNID_ATTR         SET_BIT( 14 )
#define   ULCLABEL_ATTR                       SET_BIT( 15 )
#define   ULCICON_ATTR                        SET_BIT( 16 )
```

Underlying Arc

```
#define   ULAAENDPOINTPROTOCOL_ATTR          SET_BIT( 0 )
#define   ULAZENDPOINTPROTOCOL_ATTR          SET_BIT( 1 )
#define   ULAAENDPOINTNAME_ATTR              SET_BIT( 2 )
#define   ULAZENDPOINTNAME_ATTR              SET_BIT( 3 )
#define   ULAARCINDEXID_ATTR                 SET_BIT( 4 )
#define   reserved for future use            SET_BIT( 5 )
#define   UCONNARCNAMEBINDING_ATTR           SET_BIT( 6 )
#define   UCONNAENDPOINTPROTOCOL_ATTR        SET_BIT( 7 )
#define   UCONNZENDPOINTPROTOCOL_ATTR        SET_BIT( 8 )
#define   reserved for future use            SET_BIT( 9 )
#define   UNDERLYINGARCKIND_ATTR             SET_BIT( 10 )
#define   UCONNAENDPOINTNAME_ATTR            SET_BIT( 11 )
#define   UCONNZENDPOINTNAME_ATTR            SET_BIT( 12 )
#define   UCONNARCINDEXID_ATTR               SET_BIT( 13 )
#define   reserved for future use            SET_BIT( 14 )
#define   ARCNEXTSERIALNAMEBINDING_ATTR      SET_BIT( 15 )
#define   NEXTSERIALAENDPOINTPROTOCOL_ATTR   SET_BIT( 16 )
#define   NEXTSERIALAENDPOINTNAME_ATTR       SET_BIT( 17 )
#define   NEXTSERIALZENDPOINTPROTOCOL_ATTR   SET_BIT( 18 )
#define   NEXTSERIALZENDPOINTNAME_ATTR       SET_BIT( 19 )
#define   NEXTSERIALARCINDEXID_ATTR          SET_BIT( 20 )
#define   ULALABEL_ATTR                      SET_BIT( 21 )
#define   ULAICON_ATTR                       SET_BIT( 22 )
```

Members

```
#define   MEMBERPROTOCOL_ATTR                SET_BIT( 0 )
#define   MEMBERNAME_ATTR                    SET_BIT( 1 )
#define   reserved for future use            SET_BIT( 2 )
#define   reserved for future use            SET_BIT( 3 )
#define   MEMBERNAMEBINDING_ATTR             SET_BIT( 4 )
#define   MEMBERCOMPONENTPROTOCOL_ATTR       SET_BIT( 5 )
#define   MEMBERCOMPONENTNAME_ATTR           SET_BIT( 6 )
#define   MEMBERLABEL_ATTR                   SET_BIT( 7 )
#define   MEMBERICON_ATTR                    SET_BIT( 8 )
```

Member Arcs

```
#define   MAGRAPHPROTOCOL_ATTR               SET_BIT( 0 )
#define   MAGRAPHNAME_ATTR                   SET_BIT( 1 )
#define   reserved for future use            SET_BIT( 2 )
#define   reserved for future use            SET_BIT( 3 )
#define   MANAMEBINDING_ATTR                 SET_BIT( 4 )
#define   MAAENDPOINTPROTOCOL_ATTR           SET_BIT( 5 )
#define   MAAENDPOINTNAME_ATTR               SET_BIT( 6 )
#define   MAZENDPOINTPROTOCOL_ATTR           SET_BIT( 7 )
#define   MAZENDPOINTNAME_ATTR               SET_BIT( 8 )
#define   MAARCINDEXID_ATTR                  SET_BIT( 9 )
#define   MALABEL_ATTR                       SET_BIT( 10 )
#define   MAICON_ATTR                        SET_BIT( 11 )
```

Attached Arcs

```
#define   AAGRAPHPROTOCOL_ATTR              SET_BIT( 0 )
#define   AAGRAPHNAME_ATTR                  SET_BIT( 1 )
#define   reserved for future use           SET_BIT( 2 )
#define   reserved for future use           SET_BIT( 3 )
#define   AANAMEBINDING_ATTR                SET_BIT( 4 )
#define   AAAENDPOINTPROTOCOL_ATTR          SET_BIT( 5 )
#define   AAAENDPOINTNAME_ATTR              SET_BIT( 6 )
#define   AAZENDPOINTPROTOCOL_ATTR          SET_BIT( 7 )
#define   AAZENDPOINTNAME_ATTR              SET_BIT( 8 )
#define   AAARCINDEXID_ATTR                 SET_BIT( 9 )
```

Additional Members

```
#define   AMEMBERPROTOCOL_ATTR              SET_BIT( 0 )
#define   AMEMBERNAME_ATTR                  SET_BIT( 1 )
#define   reserved for future use           SET_BIT( 2 )
#define   reserved for future use           SET_BIT( 3 )
#define   XCOORDINATE_ATTR                  SET_BIT( 4 )
#define   YCOORDINATE_ATTR                  SET_BIT( 5 )
#define   XGRID_ATTR                        SET_BIT( 6 )
#define   YGRID_ATTR                        SET_BIT( 7 )
#define   ACOMPONENTNAMEBINDING_ATTR        SET_BIT( 8 )
#define   ACOMPONENTMEMBERPROTOCOL_ATTR     SET_BIT( 9 )
#define   ACOMPONENTMEMBERNAME_ATTR         SET_BIT( 10 )
```

Additional Graph

```
#define   ADDGRAPHPROTOCOL_ATTR             SET_BIT( 0 )
#define   ADDGRAPHNAME_ATTR                 SET_BIT( 1 )
#define   ADDGRAPHINDEXID_ATTR              SET_BIT( 2 )
#define   reserved for future use           SET_BIT( 3 )
#define   reserved for future use           SET_BIT( 4 )
#define   GRAPHDESC1_ATTR                   SET_BIT( 5 )
#define   GRAPHDESCRX_ATTR                  SET_BIT( 6 )
#define   GRAPHDESCRY_ATTR                  SET_BIT( 7 )
#define   GRAPHROOTPROTOCOL_ATTR            SET_BIT( 8 )
#define   GRAPHROOTNAME_ATTR                SET_BIT( 9 )
#define   ROOTGRAPHLABEL_ATTR               SET_BIT( 10 )
#define   ROOTGRAPHICON_ATTR                SET_BIT( 11 )
```

Sample

To use a nvotVertex structure where the attributes vertexProtocol and vertexName are filled, we should set validAttributes to 0003. (set bits 0 and 1).

nvotOperationType

```
typedef enum {
  CREATE_OPERATION = 1,
  DELETE_OPERATION = 2,
  AVC_OPERATION    = 3,
  SC_OPERATION     = 4,
  GET_OPERATION    = 6
} nvotOperationType;
```

nvotNameBindingType

```
typedef enum {
  GRAPH_NAME_BINDING              = 0,
  VERTEX_NAME_BINDING             = 1,
  SIMPLE_CONN_GRAPH_NAME_BINDING  = 2,
  SIMPLE_CONN_VERTEX_NAME_BINDING = 3,
  ARC_GRAPH_GRAPH_NAME_BINDING    = 4,
  ARC_GRAPH_VERTEX_NAME_BINDING   = 5,
  ARC_VERTEX_GRAPH_NAME_BINDING   = 6,
  ARC_VERTEX_VERTEX_NAME_BINDING  = 7,
  DONT_CARE_NAME_BINDING          = 13
} nvotNameBindingType;
```

nvotLayoutType

```
typedef enum    {
  NONE_LAYOUT                = 1,
  USER_DEFINED_LAYOUT        = 2,
  POINT_TO_POINT_LAYOUT      = 3,
  BUS_LAYOUT                 = 4,
  STAR_LAYOUT                = 5,
  SPOKED_RING_LAYOUT         = 6,
  ROWCOL_LAYOUT              = 7,
  POINT_TO_POINT_RING_LAYOUT = 8,
  TREE_LAYOUT                = 9
} nvotLayoutType;
```

nvotStatusType

```
typedef enum {
  STATUS_NORMAL    = 1,
  STATUS_CRITICAL  = 2,
  STATUS_UNKNOWN   = 3,
  STATUS_MARGINAL  = 4,
  STATUS_UNMANAGED = 5
} nvotStatusType
```

nvotGraphType

```
typedef enum {
  OTHER_GRAPH   = 1,
  INVALID_GRAPH = 2,
  GRAPH         = 3,
  BOX           = 4
} nvotGraphType;
```

nvotBooleanType

```
typedef unsigned int nvotBooleanType; /* FALSE = 0, TRUE = 1 */
```

nvotOctetString

```
typedef struct {
  char * octetString,
  int octetLength;
} nvotOctetString;
```

nvotOperationalStateType

```
typedef enum {
  DISABLED = 1,
  ENABLED  = 2
} nvotOperationalStateType;
```

nvotUnknownStatusType

```
typedef nvotBooleanType nvotUnknownStatusType;
```

nvotAvailabilityStatusType

```
typedef enum {
  EMPTY_SET_AVAI_ST = 0,
  IN_TEST           = 1,
  FAILED            = 2,
  POWER_OFF         = 4,
  OFF_LINE          = 8,
  OFF_DUTY          = 16,
  DEPENDENCY        = 32,
  DEGRADED          = 64,
  NOT_INSTALLED     = 128
} nvotAvailabilityStatusType;
```

nvotAlarmStatusType

```
typedef enum {
  EMPTY_SET_ALARM_ST = 0,
  UNDER_REPAIR       = 1,
  CRITICAL           = 2,
  MAJOR              = 4,
  MINOR              = 8,
  ALARM_OUTSTANDING  = 16
} nvotAlarmStatusType;
```

nvotOwnerType

```
typedef enum {
  MINE     = 1,
  NOT_MINE = 2
} nvotOwnerType;
```

nvotUnderlyingKindType

```
typedef enum {
  SERIAL   = 1,
  PARALLEL = 2
} nvotUnderlyingKindType;
```

nvotServiceType

```
typedef enum {
  USING     = 1,
  PROVIDING = 2
} nvotServiceType;
```

nvotPositionType

```
typedef struct {
  int xCoordinate;
  int yCoordinate;
  int xGrid;
  int yGrid;
} nvotPositionType;
```

nvotReturnCode

```
typedef unsigned int nvotReturnCode;
```

Glossary and Bibliography

Glossary 1079

Bibliography 1107
NetView for AIX Publications 1107
IBM RISC System/6000 Publications 1107
NetView Publications 1108
TCP/IP Publications for AIX (RS/6000, PS/2, RT, 370) 1108
AIX SNA Services/6000 Publications 1108
Internet Request for Comments (RFCs) 1108
Related Publications 1109
AIX Trouble Ticket/6000 Publications 1109
Service Point Publication 1109
Other IBM TCP/IP Publications 1109
SNMP Information 1109
X Window System Publications 1110
X/Open Specification 1110
OSF/Motif Publications 1110
ISO/IEC Standards 1110

Glossary

This glossary includes terms and definitions from:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The ANSI/EIA Standard—440-A, *Fiber Optic Terminology*. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The Network Working Group Request for Comments: 1208.
- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.
- The *Object-Oriented Interface Design: IBM Common User Access Guidelines*, Carmel, Indiana: Que, 1992.

The following cross-references are used in this glossary:

Contrast with: This refers to a term that has an opposed or substantively different meaning.

Synonym for: This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.

Synonymous with: This is a backward reference from a defined term to all other terms that have the same meaning.

See: This refers the reader to multiple-word terms that have the same last word.

See also: This refers the reader to terms that have a related, but not synonymous, meaning.

Deprecated term for: This indicates that the term should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

A

abstract syntax. A data specification that includes all distinctions that are needed in data transmissions, but that omits (abstracts) other details such as those that depend on specific computer architectures. See also *abstract syntax notation 1 (ASN.1)* and *basic encoding rules (BER)*.

abstract syntax notation 1 (ASN.1). The Open Systems Interconnection (OSI) method for abstract syntax specified in ISO 8824. See also *basic encoding rules (BER)*.

ACB name. (1) The label of an ACB macroinstruction used to name the ACB. (2) A name specified either on the VTAM APPL definition statement or on the VTAM application program's ACB macroinstruction. Contrast with *network name*.

accelerator. (1) In CUA architecture, a key or combination of keys that invokes an application-defined function. (2) In the AIXwindows Toolkit, a keyboard alternative to a mouse button action; for example, holding the <Shift> and <M> keys on the keyboard can be made to post a menu in the same way that a mouse button action does. Accelerators typically provide increased input speed and greater convenience.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method. Synonym for *VTAM*.

action. (1) An operation on a managed object, the semantics of which are defined as part of the managed object class definition. (2) In the AIX operating system, a defined task that an application performs. An action modifies the properties of an object or manipulates the object in some way.

active. (1) The state of a resource when it has been activated and is operational. (2) In the AIX operating system, pertaining to the window pane in which the text cursor is currently positioned. (3) Contrast with *inactive* and *inoperative*.

adapter. A part that electrically or physically connects a device to a computer or to another device.

address mask. For internet subnetworking, a 32-bit mask used to identify the subnetwork address bits in the host portion of an IP address. Synonymous with *subnet mask* and *subnetwork mask*.

Address Resolution Protocol (ARP). (1) In the Internet suite of protocols, the protocol that dynamically maps an IP address to an address used by a supporting metropolitan or local area network such as Ethernet or token-ring. (2) See also *Reverse Address Resolution Protocol (RARP)*.

Administrative Domain. A collection of hosts and routers, and the interconnecting networks, managed by a single administrative authority.

Advanced Function Printing (AFP). In the AS/400 system, the ability of programs to print all-points-addressable text and images.

AFP. Advanced Function Printing.

agent. (1) In systems management, a user that, for a particular interaction, has assumed an agent role. (2) An entity that represents one or more managed objects by (a) emitting notifications regarding the objects and (b) handling requests from managers for management operations to modify or query the objects. (3) A system that assumes an agent role.

agent role. In systems management, a role assumed by a user where the user is capable of performing management operations on managed objects and of emitting notifications on behalf of managed objects.

aggregate. In programming languages, a structured collection of data objects that form a data type. (I)

aggregate resource. In the NetView Graphic Monitor Facility, an object that represents a collection of real resources.

AIX. Advanced Interactive Executive.

AIX NetView Service Point. See *NetView for AIX Service Point*.

AIX NetView/6000. See *NetView for AIX*.

AIX operating system. IBM's implementation of the UNIX operating system. The RISC System/6000 system, among others, runs the AIX operating system.

AIX SystemView NetView/6000. See *NetView for AIX*.

AIXwindows Toolkit. An object-oriented collection of C language data structures and subroutines that supplement the Enhanced X-Windows Toolkit and simplify the creation of interactive client application interfaces.

alert. (1) A message sent to a management services focal point in a network to identify a problem or an impending problem. (2) In SNA management services (SNA/MS), a high priority event that warrants immediate attention.

API. Application programming interface.

APPL. Application program.

application plane. In NetView for AIX, the submap layer on which symbols of objects that are managed by at least one network or systems management application program are displayed. Symbols on the application plane are displayed without shading, which makes them appear directly against the background plane. See also *user plane*.

application program. (1) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface. See *application programming interface (API)*.

application programming interface (API). The set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

application registration file. A file created to integrate an application program into NetView for AIX by defining (a) the application program's position in the menu structure for NetView for AIX, (b) where help information is found, (c) the number and types of parameters allowed, (d) the command used to start the application program, and (e) other characteristics of the application program.

Apply. A push button that carries out the selected choices in a window without closing the window.

arc. In graphs, a curve or line segment that links two vertices.

ARP. Address Resolution Protocol.

ASCII (American National Standard Code for Information Interchange). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

ASN.1. Abstract syntax notation 1.

attaching device. Any device that is physically connected to a network and can communicate over the network. See also *station*.

attribute. (1) A characteristic that identifies and describes a managed object. The characteristic can be determined, and possibly changed, through operations on the managed object. (2) Information within a managed object that is visible at the object boundary. An attribute has a type, which indicates the range of information given by the attribute, and a value, which is within that range. (3) Variable data that is logically a part of an object and that represents a property of the object. For example, a serial number is an attribute of an equipment object.

authentication. (1) In computer security, verification of the identity of a user or the user's eligibility to access an object. (2) In computer security, verification that a message has not been altered or corrupted. (3) In computer security, a process used to verify the user of an information system or protected resources.

authentication entity. In the Simple Network Management Protocol (SNMP), the network management agent responsible for verifying that an entity is a member of the community it claims to be in. This entity is also responsible for encoding and decoding SNMP messages according to the authentication algorithm of a given community.

authentication failure. In the Simple Network Management Protocol (SNMP), a trap that may be generated by an authentication entity when a requesting client is not a member of the SNMP community.

authorization. (1) In computer security, the right granted to a user to communicate with or make use of a computer system. (T) (2) An access right. (3) The process of granting a user either complete or restricted access to an object, resource, or function.

AUTOEXEC.BAT file. In the DOS operating system, a batch file that resides in the root directory of the boot drive. AUTOEXEC.BAT contains commands that DOS executes every time the PC is booted.

B

backend. In the AIX operating system, the program that sends output to a particular device. Synonymous with *backend program*.

backend program. Synonym for *backend*.

background picture. The diagram or image that is displayed behind other symbols to show their context or relations.

background plane. In NetView for AIX, the lowest submap layer. The background plane provides the background against which symbols are displayed. A background picture can be placed in the background plane to provide a context for viewing symbols. See also *application plane* and *user plane*.

background process. (1) A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work. (2) In the AIX operating system, a mode of program execution in which the shell does not wait for program completion before prompting the user for another command. (3) Contrast with *foreground process*.

base set. (1) The set of functions, including verbs, parameters, return codes, and what-received indications, that is supported by all products that implement a particular architecture. (2) Contrast with *option set*.

basic encoding rules (BER). The rules specified in ISO 8825 for encoding data units described in abstract syntax notation 1 (ASN.1). The rules specify the encoding technique, not the abstract syntax.

Basic Input/Output System (BIOS). Code that controls basic hardware operations, such as interactions with diskette drives, hard disk drives, and the keyboard.

batch file. A file that contains a series of commands to be processed sequentially.

behavior. (1) Ideally, a collection of assertions that describe the allowed states that a managed object can assume. An assertion can be a precondition, a postcondition, or an invariant. In practice, the behavior is often an informal description of the semantics of attributes, operations, and notifications. (2) The way in which managed objects, name bindings, attributes, notifications, and operations interact with the actual resources that they model and with each other.

BER. Basic encoding rules.

Berkeley Internet Name Domain (BIND). The Berkeley implementation of the Domain Name System (DNS).

Berkeley Software Distribution (BSD). Pertaining to any of the series of UNIX specifications or implementations distributed by the University of California at Berkeley. The mnemonic “BSD” is usually followed by a number to specify the particular version of UNIX that was distributed (for example, BSD 4.3). Many vendors use BSD specifications as standards for their UNIX products.

bind. To relate an identifier to another object in a program; for example, to relate an identifier to a value, an address or another identifier, or to associate formal parameters and actual parameters. (T)

BIND. Berkeley Internet Name Domain.

BIOS. (1) Basic Input/Output System. (2) See also *NetBIOS*.

bridge. (1) A functional unit that interconnects two local area networks that use the same logical link control protocol but may use different medium access control protocols. (T) (2) A functional unit that interconnects multiple LANs (locally or remotely) that use the same logical link control protocol but that can use different medium access control protocols. A bridge forwards a frame to another bridge based on the medium access control (MAC) address. (3) In the connection of local loops, channels, or rings, the equipment and techniques used to match circuits and to facilitate accurate data transmission. (4) Contrast with *gateway* and *router*.

broadcast. Simultaneous transmission of data to more than one destination.

browse. To look at records in a file.

BSD. Berkeley Software Distribution.

buffer. (1) To allocate and schedule the use of buffers. (A) (2) A portion of storage used to hold input or output data temporarily.

bus. (1) A facility for transferring data between several devices located between two end points, only one device being able to transmit at a given moment. (T) (2) A computer configuration in which processors are interconnected in series.

bus network. (1) A local area network in which there is only one path between any two data stations and in which data transmitted by any station is concurrently available to all other stations on the same transmission medium. (2) A network configuration that provides a bidirectional transmission facility to which all nodes are attached. A sending node transmits in both directions to the ends of the bus. All nodes in the path copy the message as it passes.

Note: A bus network may be a linear network, a star network, or a tree network. In the case of a tree or star network, there is a data station at each endpoint node. There is no data station at an intermediate node; however, one or more devices such as repeaters, connectors, amplifiers, and splitters are located there. (T)

button. (1) A mechanism on a pointing device, such as a mouse, used to request or initiate an action or a process. (2) A graphical device that identifies a choice. (3) A graphical mechanism that, when selected, performs a visible action. For example, when a user clicks on a list button, a list of choices appears. (4) See *mouse button*, *push button*, *radio button*, and *spin button*.

C

cache. (1) A special-purpose buffer storage, smaller and faster than main storage, used to hold a copy of instructions and data obtained from main storage and likely to be needed next by the processor. (T) (2) A buffer storage that contains frequently accessed instructions and data; it is used to reduce access time. (3) An optional part of the directory database in network nodes where frequently used directory information may be stored to speed directory searches. (4) To place, hide, or store in a cache.

callback. In the AIX operating system, a procedure that is called if and when certain specified conditions are met.

callback registration. The identification and registration of a callback routine.

Cancel. A push button that removes a window without applying any changes made in that window.

card. In NetView for AIX, see *event card*.

cardinality. In a relational database, the number of tuples in a relation. (T) (A)

carrier sense. In a local area network, an ongoing activity of a data station to detect whether another station is transmitting. (T)

carrier sense multiple access with collision detection (CSMA/CD). A protocol that requires carrier sense and in which a transmitting data station that detects another signal while transmitting, stops sending, sends a jam signal, and then waits for a variable time before trying again. (T) (A)

catalog. (1) A directory of files and libraries, with reference to their locations. A catalog may contain other information such as the types of devices in which the files are stored, passwords, and blocking factors. (I) (A) (2) To enter information about a file or a library into a catalog. (I) (A)

CCITT. International Telegraph and Telephone Consultative Committee. This was an organization of the International Telecommunication Union (ITU). On 1 March 1993 the ITU was reorganized, and responsibilities for standardization were placed in a subordinate organization named the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-TS). “CCITT” continues to be used for recommendations that were approved before the reorganization.

channel-attached. Pertaining to devices attached to a controlling unit by cables, rather than by telecommunication lines.

check box. A square box with associated text that represents a choice. When a user selects the choice, the check box is filled to indicate that the choice is selected. The user can clear the check box by selecting the choice again, thereby deselecting the choice.

child process. In the AIX and OS/2 operating systems, a process, started by a parent process, that shares the resources of the parent process. See also *fork*.

CID. Configuration, installation, and distribution. See *CID methodology*.

CID methodology. An IBM-specified way to install and configure products on, or remove products from, remote workstations and hosts. Response files and redirected installation and configuration may be used by a CID-enabled product to eliminate or reduce user interaction with the CID-enabled product.

circuit. (1) One or more conductors through which an electric current can flow. See *physical circuit* and *virtual circuit*. (2) A logic device.

class. (1) In object-oriented design or programming, a group of objects that share a common definition and that therefore share common properties, operations, and behavior. Members of the group are called instances of the class. (2) In the AIX operating system, pertaining to the I/O characteristics of a device. System devices are classified as block or character devices.

class A network. In Internet communications, a network in which the high-order (most significant) bit of the IP address is set to 0 and the host ID occupies the three low-order octets.

class B network. In Internet communications, a network in which the two high-order (most significant and next-to-most significant) bits of the IP address are set to 1 and 0, respectively, and the host ID occupies the two low-order octets.

class C network. In Internet communications, a network in which the two high-order (most significant and next-to-most significant) bits of the IP address are both set to 1 and the next high-order bit is set to 0. The host ID occupies the low-order octet.

click. To press and release a button on a pointing device without moving the pointer off of the object or choice.

client. (1) A functional unit that receives shared services from a server. (T) (2) A user. (3) In an AIX distributed file system environment, a system that is dependent on a server to provide it with programs or access to programs.

client/server. In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

Close. A choice that removes a window and all of the windows associated with it from the workplace. For example, if a user is performing a task in a window and a message appears, or the user asks for help, both the message and the help windows disappear when the user closes the original window.

CMIP. Common Management Information Protocol.

CMIS. Common Management Information Service.

CMISE. Common Management Information Service Element.

CMOT. Common Management Information Protocol over TCP/IP.

code page. (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (2) In the Print Management Facility, a font library member that associates code points and character identifiers. A code page also identifies invalid code points. (3) A particular assignment of hexadecimal identifiers to graphic characters. (4) In AFP support, a font file that associates code points and graphic character identifiers.

code point. (1) A 1-byte code representing one of 256 potential characters. (2) In SNA management services (SNA/MS), a 1- or 2-byte value that identifies a particular meaning to the receiver of an alert so that appropriate text can be displayed.

command. A request from a terminal for the performance of an operation or the execution of a particular program.

command list. In the NetView program, a list of commands and statements designed to perform a specific function for the user. Command lists can be written in REXX or in the NetView command list language.

Common Management Information Protocol (CMIP). The OSI standard protocol defined in ISO/IEC 9596-1 for the interaction between managers and agents that use the Common Management Information Service Element (CMISE).

Common Management Information Protocol over TCP/IP (CMOT). An Internet Engineering Task Force

(IETF) specification for the use of CMIP over a TCP/IP protocol stack.

Common Management Information Service (CMIS). The set of services provided by the Common Management Information Service Element.

Common Management Information Service Element (CMISE). The particular application service element defined in ISO/IEC 9595.

Common Programming Interface for Communications (CPI-C). An evolving application programming interface (API), embracing functions to meet the growing demands from different application environments and to achieve openness as an industry standard for communications programming. CPI-C provides access to interprogram services such as (a) sending and receiving data, (b) synchronizing processing between programs, and (c) notifying a partner of errors in the communication.

communication controller. A type of communication control unit whose operations are controlled by one or more programs stored and executed in the unit. It manages the details of line control and the routing of data through a network.

communications infrastructure. In the AIX operating system, a framework of communication that consists of a postmaster, an object registration service, a startup file, communication protocols, and application programming interfaces.

community. In the Simple Network Management Protocol (SNMP), an administrative relationship between entities.

community name. In the Simple Network Management Protocol (SNMP), a string of octets identifying a community.

component. Hardware or software that is part of a functional unit.

compression. (1) The process of eliminating gaps, empty fields, redundancies, and unnecessary data to shorten the length of records or blocks. (2) Any encoding to reduce the number of bits used to represent a given message or record.

concentrator. (1) In data transmission, a functional unit that permits a common transmission medium to serve more data sources than there are channels currently available within the transmission medium. (T) (2) Any device that combines incoming messages into a single message (concentration) or extracts individual messages from the data sent in a single transmission sequence (deconcentration).

CONFIG.SYS file. In the OS/2 operating system, a file used by the base operating system that describes the devices, system parameters, and resource options of a workstation. See also *configuration file*.

configuration. (1) The manner in which the hardware and software of an information processing system are organized and interconnected. (T) (2) The devices and programs that make up a system, subsystem, or network.

configuration file. A file that specifies the characteristics of a system device or network.

connection. (1) In data communication, an association established between functional units for conveying information. (I) (A) (2) In Open Systems Interconnection architecture, an association established by a given layer between two or more entities of the next higher layer for the purpose of data transfer. (T) (3) In TCP/IP, the path between two protocol applications that provides reliable data stream delivery service. In the Internet, a connection extends from a TCP application on one system to a TCP application on another system. (4) In system communications, a line over which data can be passed between two systems or between a system and a device. (5) Synonym for *physical connection*.

connection-oriented service. A service that establishes a logical connection between two partners for the duration that they want to communicate. Data transfer takes place in a reliable, sequenced manner. Contrast with *connectionless service*.

connectionless service. A network service that treats each packet or datagram as a separate entity that contains the source address and destination address and for which no acknowledgment is returned to the originating source. Connectionless services are on a best-effort basis and do not guarantee reliable or in-sequence delivery. Contrast with *connection-oriented service*.

connector class. In NetView for AIX, an object class used for objects that connect different parts of the network and that route or switch traffic between these parts. This class includes gateways, repeaters (including multiport repeaters), and bridges. Contrast with *network class*.

container. A visual user-interface component whose specific purpose is to hold objects.

control desk. In NetView for AIX, a component of the graphical user interface (GUI) that enables the network operator to group application program instances together.

Copy. A choice that places a copy of a selected object onto the clipboard.

Corrective Service Diskette. A diskette provided by IBM to registered service coordinators for resolving user-identified problems with previously installed software. This diskette includes program updates designed to resolve problems.

CPI-C. Common Programming Interface for Communications.

cron table. In the AIX operating system, a table used to schedule application programs and processes.

Note: "Cron" is an abbreviation for "chronological."

CSMA/CD. Carrier sense multiple access with collision detection.

Cut. A choice that moves a selected object and places it onto the clipboard. The space it occupied is usually filled by the remaining object or objects in the window.

D

daemon. A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their task; others operate periodically.

data. A representation of facts or instructions in a form suitable for communication, interpretation, or processing by human or automatic means. Data include constants, variables, arrays, and character strings.

Note: Programmers make a distinction between instructions and the data they operate on; however, in the usual sense of the word, data includes programs and program instructions.

data circuit. (1) A pair of associated transmit and receive channels that provide a means of two-way data communication. (I) (2) In SNA, synonym for *link connection*. (3) See also *physical circuit* and *virtual circuit*.

Notes:

1. Between data switching exchanges, the data circuit may include data circuit-terminating equipment (DCE), depending on the type of interface used at the data switching exchange.
2. Between a data station and a data switching exchange or data concentrator, the data circuit includes the data circuit-terminating equipment at the data station end, and may include equipment similar to a DCE at the data switching exchange or data concentrator location.

data circuit-terminating equipment (DCE). In a data station, the equipment that provides the signal conversion and coding between the data terminal equipment (DTE) and the line. (I)

Notes:

1. The DCE may be separate equipment or an integral part of the DTE or of the intermediate equipment.
2. A DCE may perform other functions that are usually performed at the network end of the line.

data dictionary. A centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. It assists management, database administrators, system analysts, and application programmers in planning, controlling, and evaluating the collection, storage, and use of data.

data link control (DLC). A set of rules used by nodes on a data link (such as an SDLC link or a token ring) to accomplish an orderly exchange of information.

data set. Synonym for *file*.

data stream. (1) All information (data and control commands) sent over a data link usually in a single read or write operation. (2) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

data terminal equipment (DTE). That part of a data station that serves as a data source, data sink, or both. (I) (A)

datagram. In TCP/IP, the basic unit of information passed across the Internet environment. A datagram contains a source and destination address along with the data. An Internet Protocol (IP) datagram consists of an IP header followed by the transport layer data.

DBCS. Double-byte character set.

DCE. (1) Data circuit-terminating equipment. (2) Distributed Computing Environment.

DCF. Document Composition Facility.

decompression. The inverse of compression.

default. Pertaining to an attribute, condition, value, or option that is assumed when none is explicitly specified. (I)

Delete. A choice that removes a selected object. The space it occupied is usually filled by the remaining object or objects in the window.

demand poll. In NetView for AIX, a polling operation initiated by the user.

destination. Any point or location, such as a node, station, or a particular terminal, to which information is to be sent.

device. A mechanical, electrical, or electronic contrivance with a specific purpose.

device driver. (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD drive.

dialog box. In OSF/Motif, a collection of data fields and buttons for setting controls, selecting from lists, choosing from mutually exclusive options, entering data, and presenting the user with messages.

directory service (DS). An application service element that translates the symbolic names used by application processes into the complete network addresses used in an OSI environment. (T)

disable. To make nonfunctional.

discarded packet. A packet that is intentionally destroyed.

discovery. In data communication, the automatic detection of network topology changes (for example, new and deleted nodes or new and deleted interfaces).

discriminator. An object that enables a system to select operations and event reports relating to other managed objects. See also *event forwarding discriminator (EFD)*.

disk operating system. An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

display. (1) A visual presentation of data. (I) (A) (2) To present data visually. (I) (A) (3) Deprecated term for *panel*.

display panel. In computer graphics, a predefined display image that defines the locations and characteristics of display fields on a display surface.

Distributed Computing Environment (DCE). The Open Software Foundation (OSF) specification (or a product derived from this specification) that assists in networking. DCE provides such functions as authentication, directory service (DS), and remote procedure call (RPC).

distributed management environment (DME). A specification of the Open Software Foundation (OSF) for managing “open” systems.

Distributed Protocol Interface (DPI). An interface between a Simple Network Management Protocol (SNMP) agent and its subagents that is defined in Request for Comments (RFC) 1592.

DLC. Data link control.

DME. Distributed management environment.

DNS. Domain Name System.

Document Composition Facility (DCF). An IBM licensed program used to format input to a printer.

domain. (1) That part of a computer network in which the data processing resources are under common control. (T) (2) In SNA, see *end node domain*, *network node domain*, and *system services control point (SSCP) domain*. (3) In Open Systems Interconnection (OSI), a part of a distributed system or a set of managed objects to which a common policy applies. (4) In a database, all the possible values of an attribute or a data element. (5) See *Administrative Domain* and *domain name*.

domain name. In the Internet suite of protocols, a name of a host system. A domain name consists of a sequence of subnames separated by a delimiter character. For example, if the fully qualified domain name (FQDN) of a host system is `ralvm7.vnet.ibm.com`, each of the following is a domain name:

- `ralvm7.vnet.ibm.com`
- `vnet.ibm.com`
- `ibm.com`

Domain Name System (DNS). In the Internet suite of protocols, the distributed database system used to map domain names to IP addresses.

domain operator. In a multiple-domain network, the person or program that controls operation of resources controlled by one system services control point (SSCP). See also *network operator*.

DOS. Disk Operating System. See *IBM Disk Operating System*.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set (SBCS)*.

double-click. To press and release a button on a pointing device twice while a pointer is within the limits that the user has specified for the operating environment.

DPI. Distributed Protocol Interface.

drag. To use a pointing device to move an object. For example, a user can drag a window border to make the window larger.

drag and drop. To directly manipulate an object by moving it and placing it somewhere else using a pointing device.

DS. Directory service.

DTE. Data terminal equipment. (A)

dump. (1) To record, at a particular instant, the contents of all or part of one storage device in another storage device. Dumping is usually for the purpose of debugging. (T) (2) Data that has been dumped. (T) (3) To copy data in a readable format from main or auxiliary storage onto an external medium such as tape, diskette, or printer.

dynamic. (1) In programming languages, pertaining to properties that can only be established during the execution of a program; for example, the length of a variable-length data object is dynamic. (I) (2) Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time. (3) Contrast with *static*.

E

e-mail. Electronic mail.

echo. (1) In computer graphics, the immediate notification of the current values provided by an input device to the operator at the display console. (I) (A) (2) In data communication, a reflected signal on a communications channel. For example, on a communications terminal, each signal is displayed twice, once when entered at the local terminal and again when returned over the communications link. This allows the signals to be checked for accuracy.

echo check. (1) A check to determine the correctness of the transmission of data in which the received data are returned to the source for comparison with the originally transmitted data. (T) (2) A method of checking the accuracy of transmission of data in which the received data are returned to the sending end for comparison with the original data. (A)

EFD. Event forwarding discriminator.

electronic mail (e-mail). (1) Correspondence in the form of messages transmitted between user terminals over a computer network. (T) (2) The generation, transmission, and display of correspondence and documents by electronic means. (A)

EMS. Event management services.

enable. To make functional.

end node domain. An end node control point, its attached links, and its local LUs.

end user. A person, device, program, or computer system that utilizes a computer network for the purpose of data processing and information exchange. (T)

end-user interface (EUI). In NetView for AIX, synonym for *graphical user interface (GUI)*.

Enhanced X-Windows Toolkit. (1) In the AIX operating system, a collection of basic functions for developing a variety of application environments. Toolkit functions manage Toolkit initialization, widgets, memory, events, geometry, input focus, selections, resources, translation of events, graphics contexts, pixmaps, and errors. (2) See also *AIXwindows Toolkit* and *X Window System*.

entity. Any concrete or abstract thing of interest, including associations among things; for example, a person, object, event, or process that is of interest in the context under consideration, and about which data may be stored in a database. (T)

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. (I) (A)

error log. (1) A data set or file in a product or system where error information is stored for later access. (2) A form in a maintenance library that is used to record error information about a product or system. (3) A record of machine checks, device errors, and volume statistical data.

error record template. In the AIX operating system, a template that describes the error class, error type, error description, probable causes, recommended actions, and failure data for an error log entry.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses carrier sense multiple access with collision detection (CSMA/CD).

EUI. End-user interface.

event. (1) An occurrence of significance to a task; for example, an SNMP trap, the opening of a window or a submap, or the completion of an asynchronous operation. (2) In the NetView and NETCENTER programs, a record indicating irregularities of operation in physical elements of a network. (3) See also *event report*.

event card. In NetView for AIX, a graphical representation, resembling a card, of the information contained in an event sent by an agent to a manager reflecting a change in the status of one of the agent's managed nodes.

event filter. In NetView for AIX, a logical expression of criteria that determine which events are forwarded to the application program that registers the event filter with the event sieve agent. A filter is referred to as "simple" or "compound" depending on how it is handled by the filter editor.

event forwarding discriminator (EFD). A managed object that describes and controls the criteria used to select which event reports are sent and to whom they are sent.

event management services (EMS). In NetView for AIX, a centralized method of generating, receiving, routing, and logging network events.

event report. The unsolicited report that an event has occurred. When a managed object emits a notification, the agent uses one or more event forwarding discriminators (EFDs) to find the destinations to which the report is sent.

event sieve. In NetView for AIX, an object that is managed by the "ovesmd" daemon, which is the event sieve agent. The event sieve agent stores information about the event sieve object in a database and reads that information when the agent is started. See also *event filter* and *event forwarding discriminator (EFD)*.

exclusive submap. In NetView for AIX, a submap that is created by an application program wanting the exclusive right to control what happens in the application plane of the submap. Contrast with *shared submap*.

exec. (1) In the AIX operating system, to overlay the current process with another executable program. (2) See also *fork*.

executable symbol. In NetView for AIX, a symbol defined such that double-clicking on it causes an application program to perform an action on a set of target objects. Contrast with *explodable symbol*.

explodable symbol. In NetView for AIX, a symbol defined such that double-clicking on it or dragging and dropping it displays the child submap of the parent object that the symbol represents. Contrast with *executable symbol*.

F

fanout. A feature that allows several data terminal equipment (DTEs) to share the same modem. Only one DTE can transmit at a time.

FDDI. Fiber Distributed Data Interface.

feature. A part of an IBM product that may be ordered separately by the customer.

Fiber Distributed Data Interface (FDDI). An American National Standards Institute (ANSI) standard for a 100-megabit-per-second LAN using optical fiber cables.

field. (1) An identifiable area in a window. Examples of fields are: an entry field, into which a user can type or place text, and a field of radio button choices, from which a user can select one choice. (2) In NetView for AIX, the building block of which objects are composed. A field is characterized by a field name, a data type (integer, Boolean, character string, or enumerated value), and a set of flags that describe how the field is treated by NetView for AIX. A field can contain data only when it is associated with an object.

field registration file. In NetView for AIX, a file used to define fields for use in the object database.

FIFO. First-in-first-out. (A)

file. A named set of records stored or processed as a unit. (T) Synonymous with *data set*.

file transfer. The transfer of one or more files from one system to another over a data link.

File Transfer Protocol (FTP). In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

filter. (1) A device or program that separates data, signals, or material in accordance with specified criteria. (A) (2) In the NetView program, a function that limits the data that is to be recorded on the database and displayed at the terminal. (3) In the AIX operating system, a command that reads standard input data, modifies the data, and sends it to the display screen. (4) See also *recording filter* and *viewing filter*.

filter editor. In NetView for AIX, a part of the graphical user interface (GUI) that enables the user to define, modify, and delete filtering rules for use by application programs.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag. (1) To mark an information item for selection for further processing. (T) (2) A character that signals the occurrence of some condition, such as the end of a word. (A)

flat file. (1) A one-dimensional or two-dimensional array: a list or table of items. (2) In a relational database, synonym for *relation*. (3) A file that has no hierarchical structure.

flow control. In data communication, control of the data transfer rate. (I)

focal point (FP). In the NetView program, the focal point domain is the central host domain. It is the central control point for any management services element containing control of the network management data.

folder. A container used to organize objects.

font. A family of characters of a given size and style; for example, 9-point Helvetica. (T)

foreground process. (1) In the AIX operating system, a process that must run to completion before another command is issued to the shell. The foreground process is in the foreground process group, which is the group that receives the signals generated by a terminal. (2) Contrast with *background process*.

fork. In the AIX operating system, to create and start a child process.

FP. Focal point.

FQDN. Fully qualified domain name.

FTP. File Transfer Protocol.

fully qualified domain name (FQDN). In the Internet suite of protocols, the name of a host system that includes all of the subnames of the domain name. An example of a fully qualified domain name is `ralvm7.vnet.ibm.com`. See also *host name*.

fully qualified name. (1) In SNA, synonym for *network-qualified name*. (2) In the Internet suite of protocols, see *fully qualified domain name (FQDN)*.

G

gateway. (1) A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures. (T) (2) In the AIX operating system, an entity that operates above the link layer and translates, when required, the interface and protocol used by one network into those used by another distinct network. (3) In TCP/IP, synonym for *router*.

general topology manager (GTM). In NetView for AIX, the component that accepts information about resources that are accessed through protocols other than the Internet Protocol (IP), stores this information in a database, and displays it to the user.

generic alert. A product-independent method of encoding alert data by means of both (a) code points indexing short units of stored text and (b) textual data.

GIF. Graphical interchange format.

global character. Synonym for *pattern-matching character*.

glyph. An image, usually of a character, in a font.

GMFHS. Graphic Monitor Facility host subsystem.

graph. A set of vertices and the set of arcs that link pairs of those vertices.

graphic monitor. The graphical user interface of the NetView Graphic Monitor Facility.

Graphic Monitor Facility host subsystem (GMFHS). A NetView feature that manages configuration and status updates for non-SNA resources.

graphical interchange format (GIF). In NetView for AIX, the format used for the background pictures of a network topology map.

graphical user interface (GUI). (1) A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device. (2) In NetView for AIX, the integrating interface application program that provides the means for displaying submaps and for integrating network application programs. The graphical user interface is a single, consistent interface that enables the user to interact with multiple application programs. Synonymous with *end-user interface (EUI)*.

GTM. General topology manager.

GUI. Graphical user interface.

H

hardcopy. (1) A permanent copy of a display image generated on an output device such as a printer or plotter, and which can be carried away. (T) (2) A printed copy of machine output in a visually readable form; for example, printed reports, listings, documents, and summaries. (3) Contrast with *softcopy*.

Help. A choice that gives a user access to helpful information about objects, choices, tasks, and products. A Help choice can appear on a menu bar or as a push button.

help panel. Information displayed by a system in response to a help request from a user.

hierarchy. The resource types, display types, and data types that make up the organization, or levels, in a network.

highlighting. Emphasizing a display element or segment by modifying its visual attributes. (I) (A)

home submap. In NetView for AIX, the first submap that appears when a map is opened. Each map has a home submap. When new maps are created, the home submap is the root submap.

host. (1) In the Internet suite of protocols, an end system. The end system can be any workstation; it does not have to be a mainframe. (2) See *host processor*.

host name. In the Internet suite of protocols, the name given to a machine. Sometimes, "host name" is used to mean *fully qualified domain name (FQDN)*; other times, it is used to mean the most specific subname of a fully qualified domain name. For example, if `ralvm7.vnet.ibm.com` is the fully qualified domain name, either of the following may be considered the host name:

- `ralvm7.vnet.ibm.com`
- `ralvm7`

host processor. (1) A processor that controls all or part of a user application network. (T) (2) In a network, the processing unit in which the data communication access method resides.

hypertext. A way of presenting information online with connections (called hypertext links) between one piece of information and another.

I

I/O. Input/output.

IAB. Internet Architecture Board.

IBM Disk Operating System (DOS). A disk operating system based on MS-DOS that operates with all IBM personal computers.

IBM Operating System/2 (OS/2). An IBM licensed program that can be used as the operating system for personal computers. The OS/2 licensed program can perform multiple tasks at the same time.

ICMP. Internet Control Message Protocol.

icon. (1) A graphic symbol, displayed on a screen, that a user can point to with a device such as a mouse in order to select a particular function or software application. (T) (2) A graphical representation of an object, consisting of an image, image background, and a label.

ID. (1) Identifier. (2) Identification.

IEEE. Institute of Electrical and Electronics Engineers.

IETF. Internet Engineering Task Force.

inactive. (1) Not operational. (2) Pertaining to a node or device not connected or not available for connection to another node or device. (3) In the AIX operating system, pertaining to a window that does not have an input focus. (4) Contrast with *active*. (5) See also *inoperative*.

Information/Management. A feature of the Information/System licensed program that provides interactive systems management applications for problem, change, and configuration management.

Information/System. An interactive retrieval program with related utilities designed to provide systems programmers with keyword access to selected technical information contained in either of its companion products, Information/MVS or Information/VM-VSE.

initial program load (IPL). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs.

inoperative. (1) The condition of a resource that has been active but is not currently active. A resource may be inoperative for reasons such as the following: (a) it may have failed, (b) it may have received an INOP

request, or (c) it may be suspended while a reactivate command is being processed. (2) See also *inactive*.

input/output (I/O). (1) Pertaining to input, output, or both. (A) (2) Pertaining to a device, process, or channel involved in data input, data output, or both.

instance. In the AIX operating system, a concrete realization of an abstract object class. An instance of a widget or a gadget is a specific data structure that contains detailed appearance and behavioral information that is used to generate a specific graphical object on-screen at run time.

instantiate. (1) To make an instance of; to replicate. (2) In object-oriented programming, to represent a class abstraction with a concrete instance of the class.

Institute of Electrical and Electronics Engineers (IEEE). A professional society accredited by the American National Standards Institute (ANSI) to issue standards for the electronics industry.

interface. A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (T)

International Organization for Standardization (ISO). An organization of national standards bodies from various countries established to promote development of standards to facilitate international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

International Telecommunication Union (ITU). The specialized telecommunication agency of the United Nations, established to provide standardized communication procedures and practices, including frequency allocation and radio regulations worldwide.

internet. A collection of networks interconnected by a set of routers that allow them to function as a single, large network. See also *Internet*.

Internet. The internet administered by the Internet Architecture Board (IAB), consisting of large national backbone networks and many regional and campus networks all over the world. The Internet uses the Internet suite of protocols.

Internet address. See *IP address*.

Internet Architecture Board (IAB). The technical body that oversees the development of the Internet suite of protocols that are known as TCP/IP.

Internet Control Message Protocol (ICMP). The protocol used to handle errors and control messages in the Internet Protocol (IP) layer. Reports of problems and incorrect datagram destinations are returned to the original datagram source. ICMP is part of the Internet Protocol.

Internet Engineering Task Force (IETF). The task force of the Internet Architecture Board (IAB) that is responsible for solving the short-term engineering needs of the Internet.

internet object. In NetView for AIX, a node or a network that can be addressed by the Internet Protocol (IP).

Internet Protocol (IP). A connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical network. However, this protocol does not provide error recovery and flow control and does not guarantee the reliability of the physical network.

Internet router. A device that enables an internet host to act as a gateway for routing data between separate networks that use a specific adapter.

Internet suite of protocols. A set of protocols developed for use on the Internet and published as Requests for Comments (RFCs).

Internetwork Packet Exchange (IPX). The network protocol used to connect Novell's servers, or any workstation or router that implements IPX, with other workstations. Although similar to the Internet Protocol (IP), IPX uses different packet formats and terminology.

interprocess communication (IPC). (1) In the AIX operating system, the process by which programs communicate data to each other and synchronize their activities. Semaphores, signals, and internal message queues are common methods of interprocess communication. (2) In the Enhanced X-Windows Toolkit, a communication path. See also *client*.

IP. Internet Protocol.

IP address. The 32-bit address defined by the Internet Protocol, standard 5, Request for Comment (RFC) 791. It is usually represented in dotted decimal notation.

IPC. Interprocess communication.

IPL. Initial program load.

IPX. Internetwork Packet Exchange.

ISO. International Organization for Standardization.

ITU. International Telecommunication Union.

ITU-T. See *ITU-TS*.

ITU-TS. International Telecommunication Union - Telecommunication Standardization Sector. The part of the International Telecommunication Union (ITU) that is responsible for developing recommendations for telecommunications.

K

keyword. (1) In programming languages, a lexical unit that, in certain contexts, characterizes some language construct; for example, in some contexts, IF characterizes an if-statement. A keyword normally has the form of an identifier. (I) (2) One of the predefined words of an artificial language. (A) (3) A name or symbol that identifies a parameter. (4) The part of a command operand that consists of a specific character string (such as `DSNAME=`).

L

LAN. Local area network.

layout. See *layout algorithm*.

layout algorithm. A method of arranging displayed or printed data.

link. The combination of the link connection (the transmission medium) and two link stations, one at each end of the link connection. A link connection can be shared among multiple links in a multipoint or token-ring configuration.

list button. A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

LLC. Logical link control.

load file generator. A Network Configuration Application/MVS function that converts Network Configuration Application configuration data to RODM load utility statements. These statements, when run through the RODM load utility, can create, update, and delete RODM objects that can be viewed through the NetView Graphic Monitor Facility (NGMF).

local. (1) Pertaining to a device accessed directly without use of a telecommunication line. (2) Contrast with *remote*.

local address. In SNA, an address used in a peripheral node in place of a network address and transformed to or from a network address by the boundary function in a subarea node.

local area network (LAN). (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T) (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network. (3) See also *Ethernet* and *token ring*. (4) Contrast with *metropolitan area network (MAN)* and *wide area network (WAN)*.

local host. (1) In TCP/IP, the host on the network at which a particular operator is working. (2) In an internet, the host to which a user's terminal is connected without using the internet.

local registration file (LRF). In NetView for AIX, a file that provides information about an agent or daemon, such as the name, the location of the executable code, the names of processes dependent on the agent or daemon, and details about the objects that an agent manages.

logical link control (LLC). The data link control (DLC) LAN sublayer that provides two types of DLC operation for the orderly exchange of information. The first type is connectionless service, which allows information to be sent and received without establishing a link. The LLC sublayer does not perform error recovery or flow control for connectionless service. The second type is connection-oriented service, which requires establishing a link prior to the exchange of information. Connection-oriented service provides sequenced information transfer, flow control, and error recovery.

LRF. Local registration file.

LU. Logical unit.

M

MAC. Medium access control.

mainframe. A computer, usually in a computer center, with extensive capabilities and resources to which other computers may be connected so that they can share facilities. (T)

MAN. Metropolitan area network.

managed node. In Internet communications, a workstation, server, or router that contains a network management agent. In the Internet Protocol (IP), the managed node usually contains a Simple Network Management Protocol (SNMP) agent.

managed object. (1) A component of a system that can be managed by a management application. (2) The systems management view of a resource that can be managed through the use of systems management protocols.

managed object class. An identified set of managed objects sharing (a) the same identified sets of attributes, notifications, and management operations (packages) and (b) the same conditions for presence of those packages.

Management Information Base (MIB). (1) A collection of objects that can be accessed by means of a network management protocol. (2) A definition for management information that specifies the information available from a host or gateway and the operations allowed. (3) In OSI, the conceptual repository of management information within an open system.

management region. In NetView for AIX, the set of managed objects on a particular map that defines the extent of the network that is being actively managed. The management region may vary across maps.

management services (MS). (1) One of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, configuration management, and change management. (2) Services that assist in the management of systems and networks in areas such as problem management, performance management, business management, operations management, configuration management, and change management.

management services focal point (MSFP). For any given management services discipline (for example, problem determination or response time monitoring), the control point that is responsible for that type of network management data for a sphere of control. This responsibility may include collecting, storing or displaying the data or all of these. (For example, a problem determination focal point is a control point that collects, stores, and displays problem determination data.)

manager. (1) In systems management, a user that, for a particular interaction, has assumed a manager role. (2) An entity that monitors or controls one or more managed objects by (a) receiving notifications regarding the objects and (b) requesting management operations to modify or query the objects. (3) A system that assumes a manager role.

manager role. In systems management, a role assumed by a user where the user is capable of issuing management operations and of receiving notifications.

map. In NetView for AIX, a database represented by a set of related submaps that provide a graphical and hierarchical presentation of a network and its systems.

mean time between failures (MTBF). For a stated period in the life of a functional unit, the mean value of the lengths of time between consecutive failures under stated conditions. (I) (A)

mean time to recovery (MTTR). For a stated period in the life of a functional unit, the average time required for corrective maintenance. Synonymous with *mean time to repair*. (T)

mean time to repair. Synonym for *mean time to recovery*. (T)

medium. A physical material in or on which data may be represented.

medium access control (MAC). In LANs, the sublayer of the data link control layer that supports medium-dependent functions and uses the services of the physical layer to provide services to the logical link control (LLC) sublayer. The MAC sublayer includes the method of determining when a device has access to the transmission medium.

menu. (1) A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated. (T) (2) A list of choices that can be applied to an object. A menu can contain choices that are not available for selection in certain contexts. Those choices are indicated by reduced contrast.

menu bar. (1) The area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus. (2) In the AIX operating system, a rectangular area at the top of the client area of a window that contains the titles of the standard pull-down menus for that application.

message. (1) An assembly of characters and sometimes control codes that is transferred as an entity from an originator to one or more recipients. A message consists of two parts: envelope and content. (T) (2) A communication sent from a person or program to another person or program.

message unit (MU). In SNA, the unit of data processed by any layer; for example, a basic information unit (BIU), a path information unit (PIU), or a request/response unit (RU).

metropolitan area network (MAN). A network formed by the interconnection of two or more networks which may operate at higher speed than those networks, may cross administrative boundaries, and may use multiple access methods. (T) Contrast with *local area network (LAN)* and *wide area network (WAN)*.

MIB. (1) MIB module. (2) Management Information Base.

MIB application program. A systems management application program used to monitor network devices.

MIB module. In the Simple Network Management Protocol (SNMP), a collection of objects relating to a common management area. See also *Management Information Base (MIB)* and *MIB variable*.

MIB object. Synonym for *MIB variable*.

MIB tree. In the Simple Network Management Protocol (SNMP), the structure of the Management Information Base (MIB).

MIB variable. In the Simple Network Management Protocol (SNMP), a specific instance of data defined in a MIB module. Synonymous with *MIB object*.

MIB view. In the Simple Network Management Protocol (SNMP), the collection of managed objects, known to the agent, that is visible to a particular community.

MIB walking. In the Simple Network Management Protocol (SNMP), a technique of looking for Management Information Base (MIB) tree information when it is presented in a hierarchical format.

migration. The installation of a new version or release of a program to replace an earlier version or release.

mnemonic. A single character of a menu item or a button label, often the first letter, that represents a function and can be typed to select that menu item or button. The mnemonic is usually shown as the underlined character.

modem (modulator/demodulator). (1) A functional unit that modulates and demodulates signals. One of the functions of a modem is to enable digital data to be transmitted over analog transmission facilities. (T) (A) (2) A device that converts digital data from a computer to an analog signal that can be transmitted on a telecommunication line, and converts the analog signal received to data for the computer.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine. (A)

monitor. (1) A device that observes and records selected activities within a data processing system for analysis. Possible uses are to indicate significant departure from the norm, or to determine levels of utilization of particular functional units. (T) (2) Software or hardware that observes, supervises, controls, or verifies operations of a system. (A)

Motif. See *OSF/Motif*.

mouse. A commonly used pointing device, containing one or more buttons, with which a user can interact with a product or the operating environment.

mouse button. A mechanism on a mouse pointing device used to select objects or choices, initiate actions, or directly manipulate objects; a user presses a mouse button to interact with a computer system. The button makes a “clicking” sound when pressed and released.

MS. Management services.

MSFP. Management services focal point.

MTBF. Mean time between failures. (I) (A)

MTTR. Mean time to repair. (I) (A)

MU. Message unit.

Multiple Virtual Storage (MVS). See *MVS*.

multipoint connection. A connection established for data transmission among more than two data stations. (I) (A)

Note: The connection may include switching facilities.

multipoint line. (1) A telecommunication line or circuit that connects two or more stations. (2) Contrast with *point-to-point line*.

multipoint network. (1) A network in which there are precisely two endpoint nodes, any number of intermediate nodes, and only one path between any two nodes. (T) (2) In data communication, a configuration in which more than two terminal installations are connected. The network may include switching facilities.

multiport repeater. A repeater that contains multiple ports, for example, ThinLAN hubs or EtherTwist hubs.

MVS. Multiple Virtual Storage. Implies MVS/370, the MVS/XA product, and the MVS/ESA product.

MVS/ESA product. Multiple Virtual Storage/Enterprise Systems Architecture.

MVS/XA product. Multiple Virtual Storage/Extended Architecture, consisting of MVS/System Product Version 2 and the MVS/XA Data Facility Product, operating on a System/370 processor in the System/370 extended architecture mode. MVS/XA allows virtual storage addressing to 2 gigabytes.

MVS/370. Multiple Virtual Storage/System Product Version 1.

N

native network. The subnetwork whose network identifier a node uses for its own network-qualified resource names.

NAU. (1) Network accessible unit. (2) Network addressable unit.

navigate. In the NetView Graphic Monitor Facility, to move between levels in the view hierarchy.

navigation tree. In NetView for AIX, a component of the graphical user interface (GUI) that displays a hierarchy of open submaps illustrating the parent-child relationship. The navigation tree enables the network operator to determine which submaps are currently open and to close, restore, or raise the windows that contain submaps.

NCP. Network Control Program.

NetBIOS. (1) Network Basic Input/Output System. A standard interface to networks, IBM personal computers (PCs), and compatible PCs, that is used on LANs to provide message, print-server, and file-server functions. Application programs that use NetBIOS do not need to handle the details of LAN data link control (DLC) protocols. (2) See also *BIOS*.

NETCENTER. A software product that assists the network operator and other technical personnel at a network control center in managing the network.

NetView Bridge. A set of application programming interfaces that allow the NetView program to interact with various types of databases in the MVS environment.

NetView command list language. An interpretive language that is unique to the NetView program and that is used to write NetView command lists in environments where REXX is not supported.

NetView for AIX. (1) Formerly known as *AIX SystemView NetView/6000* (or its abbreviated name, which is *AIX NetView/6000*). (2) An IBM licensed program for systems management in the AIX environment. NetView for AIX can use the NetView for AIX Service Point to communicate with the NetView and NETCENTER programs.

NetView for AIX Service Point. (1) Formerly known as the *AIX NetView Service Point*. (2) An IBM licensed program that operates in the AIX and UNIX environments. It functions as a gateway in an unattended environment.

NetView Graphic Monitor Facility (NGMF). A function of the NetView program that provides the network operator with a graphic topological presentation of a network controlled by the NetView program and that allows the operator to manage the network interactively.

NetView Performance Monitor (NPM). An IBM licensed program that collects, monitors, analyzes, and displays data relevant to the performance of a VTAM telecommunication network. It runs as an online VTAM application program.

NetView program. An IBM licensed program used to monitor and manage a network and to diagnose network problems.

NetView/PC. A PC-based IBM licensed program through which application programs can be used to monitor, manage, and diagnose problems in IBM Token-Ring networks, non-SNA communication devices, and voice networks.

network. (1) An arrangement of nodes and connecting branches. (T) (2) A configuration of data processing devices and software connected for information interchange. (3) A group of nodes and the links interconnecting them.

network accessible unit (NAU). A logical unit (LU), physical unit (PU), control point (CP), or system services control point (SSCP). It is the origin or the destination of information transmitted by the path control network. Synonymous with *network addressable unit*.

network address. (1) An identifier for a node, station, or unit of equipment in a network. (2) In a subarea network, an address, consisting of subarea and element fields, that identifies a link, link station, physical unit, logical unit, or system services control point. Subarea nodes use network addresses; peripheral nodes use local addresses or local-form session identifiers (LFSIDs). The boundary function in the subarea node to which a peripheral node is attached transforms local addresses or LFSIDs to network addresses and vice versa. Contrast with *network name*.

network addressable unit (NAU). Synonym for *network accessible unit*.

network administrator. A person who manages the use and maintenance of a network.

network analyzer. A network device that is programmed to monitor and analyze all traffic data that it receives on a LAN.

network class. In NetView for AIX, an object class used for symbols that represent compound objects that may contain objects such as hosts and network devices. Contrast with *connector class*.

Network Configuration Application/MVS. An IBM program offering that allows users to define and store

information about network and system resources. This information is then converted to Resource Object Data Manager (RODM) load utility statements. Network Configuration Application/MVS runs in conjunction with Information/Management. See also *load file generator*.

Network Control Program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

Network File System (NFS). A protocol developed by Sun Microsystems, Incorporated, that allows any host in a network to mount another host's file directories. Once mounted, the file directory appears to reside on the local host.

network identifier. (1) In TCP/IP, that part of the IP address that defines a network. The length of the network ID depends on the type of network class (A, B, or C). (2) A 1- to 8-byte customer-selected name or an 8-byte IBM-registered name that uniquely identifies a specific subnetwork.

network management gateway (NMG). A gateway between the NetView program, which is the SNA network management system, and the network management function of one or more non-SNA networks.

network management vector transport (NMVT). A management services request/response unit (RU) that flows over an active session between physical unit management services and control point management services (SSCP-PU session).

network manager. A program or group of programs that is used to monitor, manage, and diagnose the problems of a network.

network name. (1) The symbolic identifier by which end users refer to a network accessible unit, a link, or a link station within a given subnetwork. In APPN networks, network names are also used for routing purposes. Contrast with *network address*. (2) In a multiple-domain network, the name of the APPL statement defining a VTAM application program. The network name must be unique across domains. Contrast with *ACB name*. See *uninterpreted name*.

network node domain. An APPN network-node control point, its attached links, the network resources for which it answers directory search requests (namely, its local LUs and adjacent LEN end nodes), the adjacent APPN end nodes with which it exchanges directory search requests and replies, and other resources (such as a local storage device) associated with its own node or an adjacent end node for which it provides management services.

network operator. (1) A person who controls the operation of all or part of a network. (2) In a multiple-domain network, a person or program responsible for controlling all domains. (3) See also *domain operator*.

network-qualified name. In SNA, a name that uniquely identifies a specific resource (such as an LU or a CP) within a specific network. It consists of a network identifier and a resource name, each of which is a 1- to 8-byte symbol string. Synonymous with *fully qualified name*.

NFS. Network File System.

NGMF. NetView Graphic Monitor Facility.

NMG. Network management gateway.

NMVT. Network management vector transport.

node. (1) In network topology, the point at an end of a branch. (T) (2) The representation of a state or an event by means of a point on a diagram. (A) (3) In a tree structure, a point at which subordinate items of data originate. (A) (4) An endpoint of a link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities.

notification. (1) An unscheduled, spontaneously generated report of an event that has occurred. (2) In systems management, information emitted by a managed object relating to an event that has occurred within the managed object, such as a threshold violation or a change in configuration status.

NPM. NetView Performance Monitor.

O

object. (1) In object-oriented design or programming, an abstraction consisting of data and the operations associated with that data. See also *class*. (2) An item that a user can manipulate as a single unit to perform a task. An object can appear as text, an icon, or both.

object identifier. An administratively assigned data value of the type defined in abstract syntax notation 1 (ASN.1).

object registration service (ORS). In NetView for AIX, a component that creates and maintains a global directory of object managers, their locations, and their protocols. The postmaster daemon uses this directory to route messages and provide location transparency for managers and agents.

octet. A byte that consists of 8 bits. (T)

Off. A choice that appears in the cascaded menu from the Refresh choice. It sets the refresh function to off.

OK. A push button that accepts the information in a window and closes it. If the window contains changed information, those changes are applied before the window is closed.

On. A choice that appears in a cascaded menu from the Refresh choice. It immediately refreshes the view in a window.

online. (1) Pertaining to the operation of a functional unit when under the direct control of the computer. (T) (2) Pertaining to a user's ability to interact with a computer. (A)

online information. Information stored in a computer system that can be displayed, used, and modified in an interactive manner without any need to obtain hardcopy.

Open. A choice that leads to a window in which users can select the object they want to open.

Open Software Foundation (OSF). A nonprofit research and development organization whose goals are (a) to develop specifications and software for use in an open software environment and (b) to make the specifications and software available to information technology vendors under fair and equitable licensing terms. For example, OSF developed the Distributed Computing Environment (DCE).

Open Systems Interconnection (OSI). The interconnection of open systems in accordance with standards of the International Organization for Standardization (ISO) for the exchange of information. (T) (A)

Open Systems Interconnection (OSI) architecture. Network architecture that adheres to that particular set of ISO standards that relates to Open Systems Interconnection. (T)

Open Systems Interconnection (OSI) reference model. A model that describes the general principles of the Open Systems Interconnection, as well as the purpose and the hierarchical arrangement of its seven layers. (T)

operating system (OS). Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

operation. In object-oriented design or programming, a service that can be requested at the boundary of an object. Operations include modifying an object or disclosing information about an object.

operator. (1) A person or program responsible for managing activities controlled by a given piece of software such as MVS, the NetView program, or IMS. (2) A person who operates a device. (3) A person who keeps a system running.

option set. (1) A set of functions that may be supported by products that implement a particular architecture. A product may support any number of option sets or none. For each option set supported, all functions in that set are supported. (2) Contrast with *base set*.

ORS. Object registration service.

OS. Operating system.

OS/2 operating system. IBM Operating System/2.

OSF. Open Software Foundation.

OSF/Motif. A graphical interface that contains a toolkit, a presentation description language, a window manager, and a style guideline.

OSI. Open Systems Interconnection.

OSI management. (1) The facility to control, coordinate, and monitor the resources that allow communications to take place in the OSI environment. (2) The set of standards that are produced by ISO/IEC/CCITT for managing OSI. (3) Facilities that use some of the OSI management standards.

output. Pertaining to a device, process, or channel involved in an output process, or to the associated data or states. The word “output” may be used in place of “output data,” “output signal,” “output process,” when such a usage is clear in a given context. (T)

output device. Synonym for *output unit*.

output unit. A device in a data processing system by which data can be received from the system. (I) (A) Synonymous with *output device*.

P

packet. In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. The data, control signals, and, possibly, error control information are arranged in a specific format. (I)

packet internet groper (PING). (1) In Internet communications, a program used in TCP/IP networks to test the ability to reach destinations by sending the destinations an Internet Control Message Protocol (ICMP) echo request and waiting for a reply. (2) In communications, a test of reachability.

packet switching. The process of routing and transferring data by means of addressed packets so that a channel is occupied only during transmission of a

packet. On completion of the transmission, the channel is made available for transfer of other packets. (I)

page. (1) In a virtual storage system, a fixed-length block that has a virtual address and is transferred as a unit between real storage and auxiliary storage. (I) (A) (2) The information displayed at the same time on the screen of a display device.

panel. (1) See *window*. (2) A formatted display of information that appears on a display screen. See *help panel* and *task panel*. (3) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parallel transmission groups. Multiple transmission groups between adjacent nodes, with each group having a distinct transmission group number.

parent process. In the AIX and OS/2 operating systems, a process that creates other processes. Contrast with *child process*.

Paste. A choice that places the contents of the clipboard at the current cursor position.

path. The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

pattern-matching character. A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace a pattern-matching character. Synonymous with *global character* and *wildcard character*.

physical circuit. A circuit established without multiplexing. See also *data circuit*. Contrast with *virtual circuit*.

physical connection. (1) A connection that establishes an electrical circuit. (2) A point-to-point or multipoint connection. (3) Synonymous with *connection*.

PING. Packet internet groper.

ping command. The command that sends an Internet Control Message Protocol (ICMP) echo-request packet to a gateway, router, or host with the expectation of receiving a reply.

point-to-point. Pertaining to data transmission between two locations without the use of any intermediate display station or computer.

point-to-point connection. A connection established between two data stations for data transmission. (I) (A)

Note: The connection may include switching facilities.

point-to-point line. (1) A switched or nonswitched telecommunication line that connects a single remote station to a computer. (2) Contrast with *multipoint line*.

point-to-point network. A network arrangement made up of point-to-point links.

pointer. (1) A data element that indicates the location of another data element. (T) (2) An identifier that indicates the location of an item of data. (A)

polling. (1) On a multipoint connection or a point-to-point connection, the process whereby data stations are invited, one at a time, to transmit. (I) (2) Interrogation of devices for such purposes as to avoid contention, to determine operational status, or to determine readiness to send or receive data. (A)

pop-up menu. A menu that, when requested, appears next to the object it is associated with.

port. (1) An access point for data entry or exit. (2) A connector on a device to which cables for other devices such as display stations and printers are attached. (3) The representation of a physical connection to the link hardware. A port is sometimes referred to as an adapter; however, there can be more than one port on an adapter. There may be one or more ports controlled by a single DLC process. (4) In the Internet suite of protocols, a 16-bit number used to communicate between TCP or the User Datagram Protocol (UDP) and a higher-level protocol or application. Some protocols, such as File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP), use the same well-known port number in all TCP/IP implementations. (5) An abstraction used by transport protocols to distinguish among multiple destinations within a host machine. (6) Synonymous with *socket*.

POSIX. Portable Operating System Interface For Computer Environments. An IEEE standard for computer operating systems.

postmaster. In NetView for AIX, a process (daemon) that directs network management information between multiple application programs and agents running concurrently. The postmaster determines the route by using specified addresses or a routing table that is configured in the object registration service.

primary window. In OSF/Motif, the top-level window in an application program that can be minimized or represented by an icon. See also *submap window*.

problem determination. The process of determining the source of a problem; for example, a program component, machine failure, telecommunication facilities, user or contractor-installed programs or equipment,

environmental failure such as a power loss, or user error.

process identification number (process ID). A unique number assigned to a process by the operating system. The number is used internally by processes to communicate.

processor. In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. (T)

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM in a current unaltered release of the program.

program-to-program interface. In the NetView program, a facility that allows user programs to send data buffers to or receive data buffers from other user programs. It also allows system and application programs to send alerts to the NetView hardware monitor.

protocol. (1) A set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. (I) (2) In Open Systems Interconnection architecture, a set of semantic and syntactic rules that determine the behavior of entities in the same layer in performing communication functions. (T)

proxy agent. A process or entity that is both an agent to its manager and a manager for one or more objects. It satisfies requests from its manager by relaying those requests and translating them for the objects that it manages.

PTF. Program temporary fix.

public network. A network established and operated by a telecommunication administration or by a Recognized Private Operating Agency (RPOA) for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public. Contrast with *user-application network*.

pull-down menu. See *menu*.

push button. A button, labeled with text, graphics, or both, that represents an action that will be initiated when a user selects it.

Q

queue. (1) A list constructed and maintained so that the next data element to be retrieved is the one stored first. (T) (2) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed. (3) To arrange in or form a queue.

R

radio button. A circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which the user can select one. The circle becomes partially filled when a choice is selected.

RARP. Reverse Address Resolution Protocol.

read-only. A type of access to data that allows data to be read but not copied, printed, or modified.

real resource. In the NetView Graphic Monitor Facility, an object that represents one resource. Contrast with *aggregate resource*.

real time. (1) In Open Systems Interconnection architecture, pertaining to the processing of data by a computer in connection with another process outside the computer according to time requirements imposed by the outside process. This term is also used to describe systems operating in conversational mode and processes that can be influenced by human intervention while they are in progress. (I) (A) (2) In Open Systems Interconnection architecture, pertaining to an application such as a process control system or a computer-assisted instruction system in which response to input is fast enough to affect subsequent input.

Recognized Private Operating Agency (RPOA). Any individual, company, or corporation, other than a government department or service, that operates a telecommunication service and is subject to the obligations undertaken in the Convention of the International Telecommunication Union and in the Regulations; for example, a communication common carrier.

Recommendation X.25. See *X.25*.

recommended action. Procedures suggested by the NetView program that can be used to determine the causes of network problems.

record. (1) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (I) (2) A set of data treated as a unit. (T) (3) A set of one or more related data items grouped for processing.

recording filter. In the NetView program, the function that determines which events, statistics, and alerts are stored on a database.

redirect. To define or use a logical device name as a reference to another device or file that may be local or remote.

reduced instruction-set computer (RISC). A computer that uses a small, simplified set of frequently used instructions for rapid execution.

Refresh. A cascading choice that gives a user access to other choices (*On* and *Off*) that control whether changes made to underlying data in a window are displayed immediately, not displayed at all, or displayed at a later time.

Refresh now. A choice that shows changes made to underlying data in a window immediately.

registration file. See *application registration file*, *field registration file*, *local registration file (LRF)*, and *symbol registration file*.

relation. In a relational database, a set of entity occurrences that have the same attributes. (T)

relational database. A database in which the data are organized and accessed according to relations. (T)

remote. (1) Pertaining to a system, program, or device that is accessed through a telecommunication line. (2) Contrast with *local*.

remote procedure call (RPC). A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.

repeater. A node of a local area network, a device that regenerates signals in order to extend the range of transmission between data stations or to interconnect two branches. (T)

Request for Comments (RFC). In Internet communications, the document series that describes a part of the Internet suite of protocols and related experiments. All Internet standards are documented as RFCs.

resource. Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs.

Resource Object Data Manager (RODM). A component of the NetView program that operates as a cache manager and that supports automation applications. RODM provides an in-memory cache for maintaining real-time data in an address space that is accessible by multiple applications.

response. (1) In data communication, a reply represented in the control field of a response frame. It advises the primary or combined station of the action taken by the secondary or other combined station to one or more commands. (2) See also *command*.

response file. A file that contains a set of predefined answers to questions asked by a program and that is used in place of user dialog. See also *CID methodology*.

response time. For response time monitoring, the time from the activation of a transaction until a response is received, according to the response time definition coded in the performance class.

response time monitor (RTM). A feature available with certain hardware devices to allow measurement of response times, which may be collected and displayed by the NetView program.

Reverse Address Resolution Protocol (RARP). (1) In the Internet suite of protocols, the protocol that maps a hardware (MAC) address to an IP address. RARP can be used to determine a port's IP address. (2) See also *Address Resolution Protocol (ARP)*.

RFC. Request for Comments.

ring. See *ring network*.

ring network. (1) A network in which every node has exactly two branches connected to it and in which there are exactly two paths between any two nodes. (T) (2) A network configuration in which devices are connected by unidirectional transmission links to form a closed path.

RISC. Reduced instruction-set computer.

RODM. Resource Object Data Manager.

root user. See *superuser authority.*

route. (1) An ordered sequence of nodes and transmission groups (TGs) that represent a path from an origin node to a destination node traversed by the traffic exchanged between them. (2) The path that network traffic uses to get from source to destination.

router. (1) A computer that determines the path of network traffic flow. The path selection is made from several paths based on information obtained from specific protocols, algorithms that attempt to identify the shortest or best path, and other criteria such as metrics or protocol-specific destination addresses. (2) An attaching device that connects two LAN segments, which use similar or different architectures, at the reference model network layer. (3) In OSI terminology, a function that determines a path by which an entity can be reached. (4) In TCP/IP, synonymous with *gateway*. (5) Contrast with *bridge*.

routine. A program, or part of a program, that may have some general or frequent use. (T)

routing. (1) The process of determining the path to be used for transmission of a message over a network. (T) (2) The assignment of the path by which a message is to reach its destination.

RPC. Remote procedure call.

RPOA. Recognized Private Operating Agency.

RTM. Response time monitor.

S

SAP. Service access point.

sash. In OSF/Motif, a small square on the boundary between two components of a window, which are called "panes." The sash sets the boundary between these panes.

SBCS. Single-byte character set.

screen. (1) The physical surface of a display device upon which information is shown to users. (2) In the AIX extended curses library, a window that is as large as the display screen of the workstation. (3) Deprecated term for *display panel*.

scroll. To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen.

scroll bar. A window component that shows a user that more information is available in a particular direction and can be scrolled into view. Scroll bars can be either horizontal or vertical.

seed file. In NetView for AIX, a file that contains a list of nodes within an Administrative Domain, which the automatic discovery function uses to accelerate the generation of the network topology map.

segment. (1) A portion of a computer program that may be executed without the entire computer program being resident in main storage. (T) (2) A group of display elements. (3) A section of cable between components or devices. A segment may consist of a single patch cable, several patch cables that are connected, or a combination of building cable and patch cables that are connected. (4) In the Enhanced X-Windows Toolkit, one or more lines that are drawn but not necessarily connected at the endpoints. (5) In LANs or WANs, a subset of nodes in a network or subnet that are connected by a common physical medium.

select. To explicitly identify one or more objects to which a subsequent choice will apply.

selection. The process of explicitly identifying one or more objects to which a subsequent choice will apply.

semaphore. (1) An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file. (2) An entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. (A) (3) In the AIX operating system, an application program that usually runs in the background and is controlled by the system program controller. (4) In the Enhanced X-Windows Toolkit, a program that provides the basic windowing mechanism. It handles interprocess communication (IPC) connections from clients, demultiplexes graphics requests onto screens, and multiplexes input back to clients.

service access point (SAP). (1) In Open Systems Interconnection (OSI) architecture, the point at which the services of a layer are provided by an entity of that layer to an entity of the next higher layer. (T) (2) A logical point made available by an adapter where information can be received and transmitted. A single service access point can have many links terminating in it. (3) A logical address that allows a system to route data between a remote device and the appropriate communications support. (4) The identification of the services provided by a specific communication service provider to one of its users. For example, the Internet Protocol (IP) uses the services of a token-ring adapter. The service access point, in this case, is the name by which IP knows the adapter that is the token-ring address.

service point (SP). (1) An entry point that supports applications that provide network management for resources not under the direct control of itself as an entry point. Each resource is either under the direct control of another entry point or not under the direct control of any entry point. A service point accessing these resources is not required to use SNA sessions (unlike a focal point). A service point is needed when entry point support is not yet available for some network management function. (2) In NetView for AIX, see *NetView for AIX Service Point*.

shared. Pertaining to the availability of a resource for more than one use at the same time.

shared application program. In NetView for AIX, an application program that serves multiple action requests; however, only one instance of the application program can run in a given graphical user interface (GUI).

shared submap. In NetView for AIX, a submap on which multiple application programs manage objects on the application plane. Shared submaps allow application programs to cooperatively contribute information to the same submap. Contrast with *exclusive submap*.

shell procedure. In the AIX operating system, a series of commands, combined in a file, that carry out a particular function when the file is run or when the file is specified as a value to the SH command. Synonymous with *shell script*.

shell script. Synonym for *shell procedure*.

shutdown. The process of ending operation of a system or a subsystem, following a defined procedure.

simple connection. In NetView for AIX, the representation of connectivity as seen from one endpoint of a connection.

Simple Network Management Protocol (SNMP). In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information on devices managed is defined and stored in the application's Management Information Base (MIB).

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set (DBCS)*.

SMIT. System Management Interface Tool.

SMUX. SNMP multiplexer.

SNA. Systems Network Architecture.

SNA management services (SNA/MS). The services provided to assist in management of SNA networks.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network accessible units (NAUs), boundary function, gateway function, and intermediate session routing function components; and the transport network.

SNA/MS. SNA management services.

snapshot. In NetView for AIX, a copy of a map that reflects the topology and status of the map's nodes and links at a given moment in time.

SNMP. Simple Network Management Protocol.

SNMP multiplexer (SMUX). A protocol that is used by a subagent to provide local and remote system monitoring using the Simple Network Management Protocol (SNMP).

socket. (1) An endpoint for communication between processes or application programs. (2) Synonym for *port*.

softcopy. (1) A nonpermanent copy of the contents of storage in the form of a display image. (T) (2) One or more files that can be electronically distributed, manipulated, and printed by a user. (3) Contrast with *hardcopy*.

SP. Service point.

spin button. A component used to display, in sequence, a ring of related but mutually exclusive choices. A user can accept the value displayed in the entry field or can type a valid choice into the entry field.

SSCP. System services control point.

star network. A radial, or star-like, configuration of nodes connected to a central controller or computer in which each node exchanges data directly with the central node.

static. (1) In programming languages, pertaining to properties that can be established before execution of a program; for example, the length of a fixed length variable is static. (I) (2) Pertaining to an operation that occurs at a predetermined or fixed time. (3) Contrast with *dynamic*.

station. An input or output point of a system that uses telecommunication facilities; for example, one or more systems, computers, terminals, devices, and associated programs at a particular location that can send or receive data over a telecommunication line.

status. The condition or state of hardware or software, usually represented by a status code.

subagent. In the Simple Network Management Protocol (SNMP), something that provides an extension to the utility provided by the SNMP agent.

subarea. A portion of the SNA network consisting of a subarea node, attached peripheral nodes, and associated resources. Within a subarea node, all network accessible units (NAUs), links, and adjacent link stations (in attached peripheral or subarea nodes) that are addressable within the subarea share a common subarea address and have distinct element addresses.

submap. In NetView for AIX, a particular view of some aspect of a network that displays symbols representing objects. The application program that creates a submap determines what part of the network the submap displays.

submap pane. The area of a submap window in which the submap is displayed.

submap stack. In NetView for AIX, a component of the graphical user interface shown on the left side of each submap window. The submap stack represents the navigational path used to reach the particular submap, and it can be used to select a previously viewed submap.

submap window. In NetView for AIX, the graphical component that contains a menu bar, a submap viewing area, a status line, and a button box. A user can display multiple submap windows of an open map and an open snapshot at any given time. See also *primary window*.

subnet. (1) In TCP/IP, a part of a network that is identified by a portion of the IP address. (2) Synonym for *subnetwork*.

subnet address. In Internet communications, an extension to the basic IP addressing scheme where a portion of the host address is interpreted as the local network address.

subnet mask. Synonym for *address mask*.

subnetwork. (1) Any group of nodes that have a set of common characteristics, such as the same network ID. (2) In the AIX operating system, one of a group of multiple logical network divisions of another network, such as can be created by the Transmission Control Protocol/Internet Protocol (TCP/IP) interface program. (3) Synonymous with *subnet*.

subnetwork mask. Synonym for *address mask*.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or asynchronously with, a controlling system. (T)

subvector. A subcomponent of the network management vector transport (NMVT) major vector.

superuser authority. In the AIX operating system, the unrestricted authority to access and modify any part of the operating system, usually associated with the user who manages the system.

symbol. In NetView for AIX, a picture or an icon on a submap that represents an object (a network resource or an application). Each symbol belongs to a class, represented by the symbol's shape, and to a subclass, represented by the design within the shape. The symbol reflects characteristics of the object it represents, such as its status; it also has characteristics of its own, such as behavior.

symbol registration file. In NetView for AIX, a file used to define symbol classes and subclasses.

synchronous. (1) Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T) (2) Occurring with a regular or predictable time relationship.

System Management Interface Tool (SMIT). An interface tool of the AIX operating system for installing, maintaining, configuring, and diagnosing tasks.

system services control point (SSCP). A component within a subarea network for managing the configuration, coordinating network operator and problem determination requests, and providing directory services and other session services for end users of the network. Multiple SSCPs, cooperating as peers with one another, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its own domain.

system services control point (SSCP) domain. The system services control point, the physical units (PUs), the logical units (LUs), the links, the link stations, and all the resources that the SSCP has the ability to control by means of activation and deactivation requests.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information, that is, the end users, to be independent of and unaffected by the specific SNA network services and facilities used for information exchange.

T

TAF. Terminal access facility.

task. In a multiprogramming or multiprocessing environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. (I) (A)

task panel. Online display from which you communicate with the program in order to accomplish the program's function, either by selecting an option provided on the panel or by entering an explicit command. See also *help panel*.

TCP. Transmission Control Protocol.

TCP/IP. Transmission Control Protocol/Internet Protocol.

Telnet. In the Internet suite of protocols, a protocol that provides remote terminal connection service. It

allows users of one host to log on to a remote host and interact as directly attached terminal users of that host.

terminal. A device, usually equipped with a keyboard and a display device, that is capable of sending and receiving information.

terminal access facility (TAF). In the NetView program, a facility that allows a network operator to control a number of subsystems. In a full-screen or operator control session, operators can control any combination of such subsystems simultaneously.

TG. Transmission group.

threshold. In NetView for AIX, a setting that specifies the maximum value a statistic can reach before notification that the limit was exceeded. For example, when a monitored MIB value has exceeded the threshold, the data collector generates a threshold event.

timeout. (1) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (I) (2) A time interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

toggle. (1) Pertaining to any device having two stable states. (A) (2) Pertaining to a switching device, such as a toggle key on a keyboard, that allows a user to switch between two types of operations.

toggle button. In the AIXwindows Toolkit and the Enhanced X-Windows Toolkit, a graphical object that simulates a toggle switch; it switches sequentially from one optional state to another.

token. (1) In a local area network, the symbol of authority passed successively from one data station to another to indicate the station temporarily in control of the transmission medium. Each data station has an opportunity to acquire and use the token to control the medium. A token is a particular message or bit pattern that signifies permission to transmit. (T) (2) In LANs, a sequence of bits passed from one device to another along the transmission medium. When the token has data appended to it, it becomes a frame.

token ring. (1) According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations. (2) A FDDI or IEEE 802.5 network with a ring topology that passes tokens from one attaching ring station (node) to another. (3) See also *local area network (LAN)*.

tool palette. In NetView for AIX, a component of the graphical user interface (GUI) that enables the network operator to open application program instances by using the mouse to drag-and-drop the icons that represent the application program.

topology. In communications, the physical or logical arrangement of nodes in a network, especially the relationships among nodes and the links between them.

TP. Transaction program.

trace. A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (A)

transaction program (TP). (1) A program that processes transactions in an SNA network. There are two kinds of transaction programs: application transaction programs and service transaction programs. See also *conversation*. (2) In VTAM, a program that performs services related to the processing of a transaction. One or more transaction programs may operate within a VTAM application program that is using the VTAM application program interface (API). In that situation, the transaction program would request services from the application program, using protocols defined by that application program. The application program, in turn, could request services from VTAM by issuing the APPCCMD macroinstruction.

Transmission Control Protocol (TCP). A communications protocol used in the Internet and in any network that follows the U.S. Department of Defense standards for internetwork protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It uses the Internet Protocol (IP) as the underlying protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communications protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transmission group (TG). (1) A connection between adjacent nodes that is identified by a transmission group number. (2) In a subarea network, a single link or a group of links between adjacent nodes. When a transmission group consists of a group of links, the links are viewed as a single logical link, and the transmission group is called a *multilink transmission group (MLTG)*. A *mixed-media multilink transmission group (MMMLTG)* is one that contains links of different medium types (for example, token-ring, switched SDLC, nonswitched SDLC, and frame-relay links). (3) In an APPN network, a single link between adjacent nodes. (4) See also *parallel transmission groups*.

trap. In the Simple Network Management Protocol (SNMP), a message sent by a managed node (agent function) to a management station to report an exception condition.

tree network. A network in which there is exactly one path between any two nodes. (T)

tree structure. A data structure that represents entities in nodes, with at most one parent node for each node, and with only one root node. (T)

U

UDP. User Datagram Protocol.

underlying connection. In NetView for AIX, the representation of lower-layer connectivity that is used by higher-layer connectivity. For example, the physical connection that transports data between two IP hosts is an underlying connection.

uninterpreted name. In SNA, a character string that a system services control point (SSCP) can convert into the network name of a logical unit (LU). Typically, an uninterpreted name is used in a logon or Initiate request from a secondary logical unit (SLU) to identify the primary logical unit (PLU) with which the session is requested.

uninterruptible power supply (UPS). A buffer between utility power or other power source and a load that requires uninterrupted, precise power.

UNIX operating system. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers. The AIX operating system is IBM's implementation of the UNIX operating system.

UPS. Uninterruptible power supply.

user. (1) Any person or any thing that may issue or receive commands and messages to or from the information processing system. (T) (2) Anyone who requires the services of a computing system.

user-application network. A configuration of data processing products, such as processors, controllers, and terminals, established and operated by users for the purpose of data processing or information exchange, which may use services offered by communication common carriers or telecommunication Administrations. (T) Contrast with *public network*.

User Datagram Protocol (UDP). In the Internet suite of protocols, a protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process. UDP uses the Internet Protocol (IP) to deliver datagrams.

user exit. (1) A point in an IBM-supplied program at which a user exit routine may be given control. (2) A programming service provided by an IBM software product that may be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

user plane. In NetView for AIX, the submap layer on which symbols of objects that are not managed by an application program are displayed. Symbols on the user plane are displayed with a shadow, which makes them appear higher than symbols on the application plane. See also *background plane*.

V

value. (1) A specific occurrence of an attribute; for example, “blue” for the attribute “color.” (T) (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. (1) In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type. (I) (2) A quantity that can assume any of a given set of values. (A) (3) A name used to represent a data item whose value can be changed while the program is running. (4) In the Simple Network Management Protocol (SNMP), a match of an object instance name with an associated value.

version. A separately licensed program that usually has significant new code or new function.

vertex. In graphs, a point that may be the end of an arc or the intersection of multiple arcs.

view preprocessor. The part of the NetView Graphic Monitor Facility that creates unformatted views of SNA resources from the VTAM definition library (VTAMLST).

viewing filter. In the NetView program, the function that allows a user to select the alert data to be displayed on a terminal. All other stored data is blocked.

virtual circuit. In packet switching, the facilities provided by a network that give the appearance to the user of an actual connection. (T) See also *data circuit*. Contrast with *physical circuit*.

virtual machine (VM). (1) A virtual data processing system that appears to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system. (T) (2) In VM/ESA, the virtual processors, virtual storage, virtual devices, and virtual channel subsystem allocated to a single user. A virtual machine also includes any expanded storage dedicated to it.

Virtual Machine/Enterprise Systems Architecture (VM/ESA). An IBM licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a real machine.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VM. Virtual machine.

VM/ESA. Virtual Machine/Enterprise Systems Architecture.

VTAM. (1) Virtual Telecommunications Access Method. (2) Synonymous with *ACF/VTAM*.

W

WAN. Wide area network.

wide area network (WAN). (1) A network that provides communication services to a geographic area larger than that served by a local area network or a metropolitan area network, and that may use or provide public communication facilities. (T) (2) A data communication network designed to serve an area of hundreds or thousands of miles; for example, public and private packet-switching networks, and national telephone networks. (3) Contrast with *local area network (LAN)* and *metropolitan area network (MAN)*.

widget. (1) In the AIX operating system, a graphic device that can receive input from the keyboard or mouse and can communicate with an application or with another widget by means of a callback. Every widget is a member of only one class and always has a window associated with it. (2) The fundamental data type of the Enhanced X-Windows Toolkit. (3) An object that provides a user-interface abstraction; for example, a Scrollbar widget. It is the combination of an Enhanced X-Windows window (or subwindow) and its associated semantics. A widget implements procedures through its widget class structure.

wildcard character. Synonym for *pattern-matching character*.

window. (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area with visible boundaries that presents a view of an object or with which a user conducts a dialog with a computer system.

work space. (1) That portion of main storage that is used by a computer program for temporary storage of data. (I) (A) (2) In NetView for AIX, a container for a set of event cards that meet certain criteria. See also *event filter*.

workstation. (1) A functional unit at which a user works. A workstation often has some processing capability. (T) (2) One or more programmable or nonprogrammable devices that allow a user to do work. (3) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

write access. In computer security, permission to write to an object.

X

X Window System. A software system, developed by the Massachusetts Institute of Technology, that allows the user of a display to concurrently use multiple application programs through different windows of the display. The application programs may execute on different computers.

X.25. (1) An International Telegraph and Telephone Consultative Committee (CCITT) recommendation for the interface between data terminal equipment and packet-switched data networks. (2) See also *packet switching*.

X.25 interface. An interface consisting of a data terminal equipment (DTE) and a data circuit-terminating equipment (DCE) in communication over a link using the procedures described in the CCITT Recommendation X.25.

Z

zoom. In CUA architecture, to progressively increase or decrease the size of a part of an image on a screen or in a window.

Bibliography

NetView for AIX Publications

The following paragraphs briefly describe the publications for Version 4 of the NetView for AIX program:

NetView for AIX Concepts: A General Information Manual (GC31-8160)

This book provides an overview of the NetView for AIX program that business executives can use to evaluate the product. System planners can also use this information to learn how NetView for AIX manages heterogeneous networks.

NetView for AIX Database Guide (SC31-8167)

This book provides information for system administrators and database administrators to configure the NetView for AIX program to work with the following relational database management systems: DB2/6000, INFORMIX, INGRES, ORACLE, and SYBASE. This book also describes how to transfer IP topology, trapdlog, and snmpCollect data to the relational database and how to manipulate the data.

NetView for AIX Installation and Configuration (SC31-8163)

This book provides installation and configuration steps for the system programmer who will install and configure the NetView for AIX program.

NetView for AIX User's Guide for Beginners (SC31-8158)

This book contains “how-to” information that provides network operators the help they need to get acquainted with NetView for AIX and accomplish some basic networking tasks. It is written for the user who is unfamiliar with the NetView for AIX program.

NetView for AIX Administrator's Guide (SC31-8168)

This book explains network management principles and describes how the NetView for AIX program's components work together. It is for the advanced user. Most of the tasks require root authority. This book includes tasks such as customizing the graphical interface, filtering events, configuring events, and managing network performance and configuration.

NetView for AIX Administrator's Reference (SC31-8169)

This book contains reference information for commands, daemons, and files. It is used primarily when performing administrative tasks.

NetView for AIX Diagnosis Guide (SC31-8162)

This book is intended to help you classify and resolve problems related to the operation of the NetView for AIX program.

NetView for AIX Application Interface Style Guide (SC31-6240)

This book provides guidelines for system programmers who develop applications that will be integrated with the NetView for AIX program.

NetView for AIX Programmer's Guide (SC31-8164)

This book provides information for programmers about creating network management applications. This book also contains information about the NetView for AIX program server, commands, function calls, and object classes.

NetView for AIX Programmer's Reference (SC31-8165)

This book is intended for programmers and contains reference information about the NetView for AIX program and its server, commands, function calls, and object classes.

NetView for AIX and the Host Connection (SC31-8161)

This book provides information for System/390 and NetView users who want to manage TCP/IP and SNA networks.

Quick Reference Card (SX75-0113)

This summary provides a brief description of each NetView for AIX daemon. The card also lists the menu items and the submenu items below them.

In addition to these printed books, online documentation of the NetView for AIX library is available. An online Help Index is also available from the NetView for AIX Help pull-down window. The Help Index provides dialog box help and task help.

IBM RISC System/6000 Publications

In addition to the NetView for AIX documentation, the following publications may also be helpful to users:

AIX Quick Reference (SC23-2401)

Task Index and Glossary for IBM RISC System/6000 (GC23-2201)

IBM RISC System/6000 Problem Solving Guide (SC23-2204)

AIX Communications Concepts and Procedures for IBM RISC System/6000 (GC23-2203)

AIX Commands Reference for IBM RISC System/6000 (GC23-2366, GC23-2367, GC23-2376, GC23-2393)

AIX Files Reference for IBM RISC System/6000 (GC23-2200)

NetView Publications

The following list contains selected NetView Version 2 Release 3 publications:

NetView Administration Reference (SC31-6128)

NetView At a Glance (GC31-7016)

NetView Automation Planning (SC31-6141)

NetView Customization Guide (SC31-6132)

NetView Installation and Administration Guide (MVS: SC31-6125) (VM: SC31-6182) (VSE: SC31-6182)

NetView Operation (SC31-6127)

NetView Problem Determination and Diagnosis (LY43-0014)

NetView Resource Alerts Reference (SC31-6136)

NetView Samples (MVS: SC31-6126) (VM: SC31-6183) (VSE: SC31-6184)

The following list contains selected NetView Version 2 Release 4 publications:

NetView Administration Reference (SC31-7080)

NetView Automation Planning (SC31-7082)

NetView Customization Guide (SC31-7091)

NetView General Information (GC31-7098)

NetView Installation and Administration Facility/2 Guide (SC31-7099)

NetView Installation and Administration Guide (SC31-7084)

NetView Operation (SC31-7066)

NetView Problem Determination and Diagnosis (LY43-0101)

NetView Resource Alerts Reference (SC31-7097)

TCP/IP Publications for AIX (RS/6000, PS/2, RT, 370)

The following list shows the books available for TCP/IP in the AIX Operating System library:

AIX Operating System TCP/IP User's Guide (SC23-2309)

AIX PS/2 TCP/IP User's Guide (SC23-2047)

TCP/IP for IBM X-Windows on DOS (SC23-2349)

AIX SNA Services/6000 Publications

The following list of publications are for use with the AIX Operating System:

AIX SNA Server/6000 User's Guide (SC31-7002)

AIX SNA Server/6000 Configuration Reference (SC31-7014)

AIX SNA Server/6000 Transaction Program (SC31-7003)

Internet Request for Comments (RFCs)

The following documents describe Internet standards supported by the NetView for AIX program. Copies of these documents are shipped on the AIX SystemView NetView/6000 product installation media. They are installed in the /usr/OV/doc directory.

RFC 1095: The Common Management Services and Protocol over TCP/IP (CMOT)

RFC 1155: Structure and Identification of Management Information for TCP/IP-Based Internets

RFC 1157: Simple Network Management Protocol (SNMP)

RFC 1187: Bulk Table Retrieval with the SNMP

RFC 1189: The Common Management Information Services and Protocols for the Internet (CMOT and CMIP)

RFC 1212: Concise MIB Definitions

RFC 1213: Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II

RFC 1215: Convention for Defining Traps for Use with the SNMP

RFC 1229: Extensions to the Generic-Interface MIB

RFC 1230: IEEE 802.4 Token Bus MIB

RFC 1231: IEEE 802.5 Token Bus MIB

RFC 1232: Definitions of Managed Objects for the DS1 Interface Type

RFC 1233: Definitions of Managed Objects for the DS3 Interface Type

RFC 1239: Reassignment of Experimental MIBs to Standard MIBs

RFC 1243: AppleTalk Management Information Base

RFC 1253: OSPF Version 2 Management Information Base

RFC 1269: Definitions of Managed Objects for the Border Gateway Protocol (Version 3)

RFC 1271: Remote Network Monitoring Management Information Base

RFC 1284: Definitions of Managed Objects for the Ethernet-like Interface Types

RFC 1285: FDDI Management Information Base

RFC 1286: Definitions of Managed Objects for Bridges

RFC 1289: DECnet Phase IV MIB Extensions

RFC 1304: Definition of Managed Objects for the SIP Interface Type

RFC 1315: Management Information Base for Frame Relay DTEs

RFC 1316: Definitions of Managed Objects for Character Stream Devices

RFC 1317: Definitions of Managed Objects for RS-232-like Hardware Devices

RFC 1318: Definitions of Managed Objects for Parallel-printer-like Hardware Devices

RFC 1450: Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1452: Coexistence between Version 1 and Version 2 of the Internet-Standard Network Management Framework

Related Publications

The following publications are closely related to or referenced by the NetView for AIX Library:

AIX Trouble Ticket/6000 Publications

For information about the AIX Trouble Ticket/6000 program, consult the following publications:

AIX Trouble Ticket/6000 Brochure (GC31-7161)

AIX Trouble Ticket/6000 User's Guide (SC31-7162)

Service Point Publication

AIX NetView Service Point Installation, Operation, and Programming Guide (SC31-6120)

Other IBM TCP/IP Publications

The following list shows other available IBM TCP/IP publications:

Introducing IBM Transmission Control Protocol/Internet Protocol Products for OS/2, VM, and MVS (GC31-6080)

IBM TCP/IP Version 2 for VM and MVS: Diagnosis Guide (LY43-0013)

MVS/DFP Version 3 Release 3: Using the Network File System Server (SC26-4732)

SNMP Information

You can use the following sources for detailed SNMP information:

The Simple Book, M.T. Rose, Prentice-Hall, 1991 (ISBN 0-13-812611-9)

The *Windows SNMP Manager API Specification*, the *WinSNMP/MIB API Specification*, and other information on Windows SNMP are available through anonymous FTP from the host sunsite.unc.edu under the directory path /pub/micro/pc-stuff/ms-windows/WinSNMP

These Internet standards provide SNMP information:

RFC 1901: Introduction to Community-based SNMPv2

RFC 1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1903: Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1904: Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1905: Protocol Operation for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1906: Transport Mapping for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1907: Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)

RFC 1908: Coexistence between Version 1 and Version 2 of Internet-standard Network Management Framework

RFC 1909: An Administrative Infrastructure for SNMPv2 (SNMPv2USEC)

RFC 1910: User-based Security Model for SNMPv2 (SNMPv2USEC)

X Window System Publications

The following list shows selected X Window System publications:

Introduction to the X Window System, Oliver Jones, Prentice-Hall, 1988 (ISBN 0-13-499997)

X Window System Technical Reference, Steven Mikes, Addison-Wesley, 1990 (ISBN 0-201-52370)

X Window System: Programming and Applications with Xt, Douglas A. Young, Prentice-Hall, 1989 (ISBN 0-13-972167)

X Window System: Programming and Applications with Xt, OSF/Motif Edition, Douglas A. Young, Prentice-Hall, 1990 (ISBN 0-13-497074)

X/Open Specification

For information about the X/Open OSI-Abstract-Data Manipulation (XOM) application programming interface (API), consult the following X/Open documents:

X/Open OSI-Abstract-Data Manipulation (XOM) API, CAE Specification

X/Open Preliminary Specification. Systems Management: GDMO to XOM Translation Algorithm

OSF/Motif Publications

The following list contains selected OSF/Motif publications:

OSF/Motif Series (5 volumes), Open Software Foundation, Prentice Hall, Inc. 1990

OSF/Motif Application Environment Specifications, (AES) (ISBN 0-13-640483-9)

OSF/Motif Programmer's Guide (ISBN 0-13-640509-6)

OSF/Motif Programmer's Reference, (ISBN 0-13-640517-7)

OSF/Motif Style Guide (ISBN 0-13-640491-X)

OSF/Motif User's Guide, (ISBN 0-13-640525-8)

ISO/IEC Standards

For information about the ISO/IEC standards on which the NetView for AIX program is based, refer to the following publications:

ISO IS 7498-4, Open Systems Interconnection–Basic Reference Model–Part 4: Management Framework

ISO 8824, Open Systems Interconnection–Specification of Abstract Syntax Notation One (ASN.1)

ISO 8825, Open Systems Interconnection– Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

ISO IS 9595, Common Management Information–Service Definition

ISO IS 9596-1, Common Management Information–Protocol Specification

ISO DIS 9899, Information Processing–Programming Language C

ISO 10040, Systems Management Overview

The ISO/IEC standards can be obtained from the following address:

OMNICOM
243 Church St. NW
Vienna, VA 22180-4434

(800) OMNICOM
(703) 281-1135
(703) 281-1505 (FAX)

Communicating Your Comments to IBM

NetView for AIX
Programmer's Reference
Version 4

Publication No. SC31-8165-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States and Canada: **1-800-227-5088**
- If you prefer to send comments electronically, use this network ID:
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**
  - Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Help us help you!

NetView for AIX
Programmer's Reference
Version 4

Publication No. SC31-8165-00

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Your comments and suggestions will help us improve our technical publications. Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied
	☐	☐

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate	☐	☐
Complete	☐	☐
Easy to find	☐	☐
Easy to understand	☐	☐
Well organized	☐	☐
Applicable to your task	☐	☐

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Your Internet Address: _____

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8165-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Information Development
Department CGMD
International Business Machines Corporation
PO BOX 12195
RESEARCH TRIANGLE PARK NC 27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8165-00

Cut or Fold
Along Line

Program Number: 5765-527

Printed in U.S.A.

SC31-8165-00

DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2330 OF '.EDF#CV'
DSMMOM397I '.EDF#CV' WAS IMBEDDED AT LINE 190 OF '.EDF#FCV7'
DSMMOM397I '.EDF#FCV7' WAS IMBEDDED AT LINE 330 OF '.EDFCOVER'
DSMMOM397I '.EDFCOVER' WAS IMBEDDED AT LINE 57 OF 'LBWL0MST'
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2 OF 'LBWL0C2'
DSMMOM397I 'LBWL0C2' WAS IMBEDDED AT LINE 187 OF 'EDFPRF40'
DSMBEG323I STARTING PASS 2 OF 4.
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2330 OF '.EDF#CV'
DSMMOM397I '.EDF#CV' WAS IMBEDDED AT LINE 190 OF '.EDF#FCV7'
DSMMOM397I '.EDF#FCV7' WAS IMBEDDED AT LINE 330 OF '.EDFCOVER'
DSMMOM397I '.EDFCOVER' WAS IMBEDDED AT LINE 57 OF 'LBWL0MST'
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2 OF 'LBWL0C2'
DSMMOM397I 'LBWL0C2' WAS IMBEDDED AT LINE 187 OF 'EDFPRF40'
DSMBEG323I STARTING PASS 3 OF 4.
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2330 OF '.EDF#CV'
DSMMOM397I '.EDF#CV' WAS IMBEDDED AT LINE 190 OF '.EDF#FCV7'
DSMMOM397I '.EDF#FCV7' WAS IMBEDDED AT LINE 330 OF '.EDFCOVER'
DSMMOM397I '.EDFCOVER' WAS IMBEDDED AT LINE 57 OF 'LBWL0MST'
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2 OF 'LBWL0C2'
DSMMOM397I 'LBWL0C2' WAS IMBEDDED AT LINE 187 OF 'EDFPRF40'
DSMBEG323I STARTING PASS 4 OF 4.
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2330 OF '.EDF#CV'
DSMMOM397I '.EDF#CV' WAS IMBEDDED AT LINE 190 OF '.EDF#FCV7'
DSMMOM397I '.EDF#FCV7' WAS IMBEDDED AT LINE 330 OF '.EDFCOVER'
DSMMOM397I '.EDFCOVER' WAS IMBEDDED AT LINE 57 OF 'LBWL0MST'
DSMKPO653E POSTSCRIPT FILE '@E@P@S' NOT FOUND.
DSMMOM395I '.EDFPO' LINE 70: .po @E@P@S
DSMMOM397I '.EDFPO' WAS IMBEDDED AT LINE 910 OF '.EDFAWRK'
DSMMOM397I '.EDFAWRK' WAS IMBEDDED AT LINE 2 OF 'LBWL0C2'
DSMMOM397I 'LBWL0C2' WAS IMBEDDED AT LINE 187 OF 'EDFPRF40'