## Contents

Preface ................................................................................................................. xi
Who Should Read This Guide ............................................................................ xi
Prerequisite and Related Documents ................................................................. xi
What This Guide Contains .................................................................................. xi
Typeface Conventions ........................................................................................ xi
Online Information ............................................................................................... xii
Accessibility Information ...................................................................................... xii
   Keyboard Access ..................................................................................... xii
Contacting Tivoli Support .................................................................................... xii

### Chapter 1. Function Tables for NetView Reference Pages

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical User Interface Routines</td>
<td>1</td>
</tr>
<tr>
<td>SNMP Commands</td>
<td>10</td>
</tr>
<tr>
<td>SNMP Routines</td>
<td>10</td>
</tr>
<tr>
<td>Open Topology Functions</td>
<td>12</td>
</tr>
<tr>
<td>GTM API Routines</td>
<td>13</td>
</tr>
<tr>
<td>IP Topology Functions</td>
<td>15</td>
</tr>
<tr>
<td>Client/Server APIs</td>
<td>15</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>15</td>
</tr>
</tbody>
</table>

### Chapter 2. Reference Pages

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format of Reference Pages</td>
<td>19</td>
</tr>
<tr>
<td>addtrap</td>
<td>20</td>
</tr>
<tr>
<td>appmon</td>
<td>25</td>
</tr>
<tr>
<td>event</td>
<td>28</td>
</tr>
<tr>
<td>evtstat</td>
<td>30</td>
</tr>
<tr>
<td>findroute</td>
<td>31</td>
</tr>
<tr>
<td>graph</td>
<td>32</td>
</tr>
<tr>
<td>gtmmd</td>
<td>35</td>
</tr>
<tr>
<td>ipmap</td>
<td>37</td>
</tr>
<tr>
<td>loadhosts</td>
<td>44</td>
</tr>
<tr>
<td>loadmib</td>
<td>46</td>
</tr>
<tr>
<td>lrf</td>
<td>48</td>
</tr>
<tr>
<td>mibExpr.conf</td>
<td>51</td>
</tr>
<tr>
<td>mibform</td>
<td>53</td>
</tr>
<tr>
<td>mibserver</td>
<td>55</td>
</tr>
<tr>
<td>mibtable</td>
<td>56</td>
</tr>
<tr>
<td>mib2trap(1)</td>
<td>57</td>
</tr>
<tr>
<td>netmon</td>
<td>59</td>
</tr>
<tr>
<td>netview</td>
<td>65</td>
</tr>
<tr>
<td>netviewd</td>
<td>68</td>
</tr>
<tr>
<td>nmdemandpoll</td>
<td>69</td>
</tr>
<tr>
<td>nmpolling</td>
<td>70</td>
</tr>
<tr>
<td>nvcold</td>
<td>72</td>
</tr>
<tr>
<td>nvcoll</td>
<td>73</td>
</tr>
<tr>
<td>nvecho</td>
<td>74</td>
</tr>
<tr>
<td>nvmail</td>
<td>75</td>
</tr>
<tr>
<td>nvpage</td>
<td>76</td>
</tr>
<tr>
<td>NVgetServerName</td>
<td>78</td>
</tr>
<tr>
<td>NVinstallMode</td>
<td>79</td>
</tr>
<tr>
<td>NVIsClient</td>
<td>80</td>
</tr>
<tr>
<td>nvotChangeArcDetails</td>
<td>81</td>
</tr>
<tr>
<td>nvotChangeArcIconInGraph</td>
<td>85</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>nvotChangeArcLabelInGraph</td>
<td>89</td>
</tr>
<tr>
<td>nvotChangeArcStatus</td>
<td>94</td>
</tr>
<tr>
<td>nvotChangeBoxBackground</td>
<td>98</td>
</tr>
<tr>
<td>nvotChangeBoxDetails</td>
<td>101</td>
</tr>
<tr>
<td>nvotChangeBoxIconInGraph</td>
<td>104</td>
</tr>
<tr>
<td>nvotChangeBoxLabelInGraph</td>
<td>107</td>
</tr>
<tr>
<td>nvotChangeBoxPositionInGraph</td>
<td>110</td>
</tr>
<tr>
<td>nvotChangeGraphBackground</td>
<td>113</td>
</tr>
<tr>
<td>nvotChangeGraphDetails</td>
<td>116</td>
</tr>
<tr>
<td>nvotChangeGraphIconInGraph</td>
<td>119</td>
</tr>
<tr>
<td>nvotChangeGraphLabelInGraph</td>
<td>122</td>
</tr>
<tr>
<td>nvotChangeGraphPositionInGraph</td>
<td>125</td>
</tr>
<tr>
<td>nvotChangeRootGraphIcon</td>
<td>128</td>
</tr>
<tr>
<td>nvotChangeRootGraphLabel</td>
<td>131</td>
</tr>
<tr>
<td>nvotChangeVertexDetails</td>
<td>134</td>
</tr>
<tr>
<td>nvotChangeVertexIconInBox</td>
<td>137</td>
</tr>
<tr>
<td>nvotChangeVertexIconInGraph</td>
<td>140</td>
</tr>
<tr>
<td>nvotChangeVertexLabelInBox</td>
<td>143</td>
</tr>
<tr>
<td>nvotChangeVertexLabelInGraph</td>
<td>146</td>
</tr>
<tr>
<td>nvotChangeVertexPositionInBox</td>
<td>149</td>
</tr>
<tr>
<td>nvotChangeVertexPositionInGraph</td>
<td>152</td>
</tr>
<tr>
<td>nvotChangeVertexStatus</td>
<td>155</td>
</tr>
<tr>
<td>nvotCreateArcInGraph</td>
<td>158</td>
</tr>
<tr>
<td>nvotCreateBoxInGraph</td>
<td>164</td>
</tr>
<tr>
<td>nvotCreateGraphInGraph</td>
<td>169</td>
</tr>
<tr>
<td>nvotCreateProvidingSap</td>
<td>174</td>
</tr>
<tr>
<td>nvotCreateRootGraph</td>
<td>178</td>
</tr>
<tr>
<td>nvotCreateUsingSap</td>
<td>182</td>
</tr>
<tr>
<td>nvotCreateVertexInBox</td>
<td>185</td>
</tr>
<tr>
<td>nvotCreateVertexInGraph</td>
<td>189</td>
</tr>
<tr>
<td>nvotDeleteArc</td>
<td>193</td>
</tr>
<tr>
<td>nvotDeleteArcFromGraph</td>
<td>196</td>
</tr>
<tr>
<td>nvotDeleteBox</td>
<td>200</td>
</tr>
<tr>
<td>nvotDeleteBoxFromGraph</td>
<td>202</td>
</tr>
<tr>
<td>nvotDeleteGraph</td>
<td>205</td>
</tr>
<tr>
<td>nvotDeleteGraphFromGraph</td>
<td>207</td>
</tr>
<tr>
<td>nvotDeleteProvidingSap</td>
<td>210</td>
</tr>
<tr>
<td>nvotDeleteUsingSap</td>
<td>213</td>
</tr>
<tr>
<td>nvotDeleteVertex</td>
<td>216</td>
</tr>
<tr>
<td>nvotDeleteVertexFromBox</td>
<td>218</td>
</tr>
<tr>
<td>nvotDeleteVertexFromGraph</td>
<td>221</td>
</tr>
<tr>
<td>nvotDone</td>
<td>224</td>
</tr>
<tr>
<td>nvotFree</td>
<td>226</td>
</tr>
<tr>
<td>nvotGetArcsInGraph</td>
<td>228</td>
</tr>
<tr>
<td>nvotGetBoxesInGraph</td>
<td>231</td>
</tr>
<tr>
<td>nvotGetError</td>
<td>234</td>
</tr>
<tr>
<td>nvotGetErrorMsg</td>
<td>237</td>
</tr>
<tr>
<td>nvotGetGraphsInGraph</td>
<td>238</td>
</tr>
<tr>
<td>nvotGetSapsOnVertex</td>
<td>241</td>
</tr>
<tr>
<td>nvotGetVerticesInBox</td>
<td>244</td>
</tr>
<tr>
<td>nvotGetVerticesInGraph</td>
<td>247</td>
</tr>
<tr>
<td>nvotInit</td>
<td>250</td>
</tr>
<tr>
<td>nvotSetCenterBoxForGraph</td>
<td>253</td>
</tr>
<tr>
<td>nvotSetCenterGraphForGraph</td>
<td>256</td>
</tr>
<tr>
<td>nvotSetSynchronousCreation</td>
<td>259</td>
</tr>
<tr>
<td>nvplay</td>
<td>261</td>
</tr>
</tbody>
</table>
Chapter 3. Using NetView GTM Data Structures

Basic Structures
- Vertex
- Graph
- Box
- Arc
- Sap

Table Structures
- nvotVertexAttrType
- nvotGraphAttrType
- nvotArcAttrType
- nvotSapAttrType

Type Structures
- nvotVertexProtocolType
- nvotGraphProtocolType
- nvotProtocolType
- nvotTableType
- nvotAttributeBitmapType
- Vertex
- Graph
- Arc
- Sap
- Sample
- nvotOperationType
- nvotNameBindingType
- nvotLayoutType
nvotStatusType ........................................ 719
nvotGraphType ........................................ 719
nvotBooleanType ........................................ 719
nvotOctetString ........................................ 719
nvotOperationalStateType ......................... 719
nvotUnknownStatusType ......................... 719
nvotAvailabilityStatusType ................... 719
nvotAlarmStatusType ................................ 719
nvotOwnerType ........................................ 720
nvotServiceType ........................................ 720
nvotPositionType ........................................ 720
nvotReturnCode ........................................ 720

Index .................................................. 721
Preface

This document provides function tables, command syntax and general topology manager data structures. This book should be used with the Tivoli® NetView for Windows® Programmer's Guide.

Who Should Read This Guide

If you are customizing the Tivoli NetView for Windows program or are writing and customizing network management applications that are to be used through the NetView program, refer to this book for descriptions of commands, daemons, files, and API functions. Programmers should have programming experience in the following areas:

- C-programming language
- Data communications
- Networking
- Windows operating system

Prerequisite and Related Documents

The following is a list of Tivoli NetView for Windows related publications:

- Tivoli NetView for Windows Programmer's Guide
- Tivoli NetView for Windows User's Guide

What This Guide Contains

The Tivoli NetView for Windows Programmer's Reference contains the following information:

- "Chapter 1. Function Tables for NetView Reference Pages" on page 1
  Contains tables that provide information about the NetView reference pages.
- "Chapter 2. Reference Pages" on page 19
  Contains the reference pages for the NetView program, organized alphabetically.
- "Chapter 3. Using NetView GTM Data Structures" on page 713
  Describes the data structures used by the general topology manager (GTM) application programming interface (API) and defined in the nvotTypes.h file.

The glossary at the end of this document can assist you with terminology. To view additional terminology lists, refer to:


Typeface Conventions

This guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

**Bold** Commands, keywords, file names, authorization roles, URLs, or other information that you must use literally appear in **bold** print. The names of titles of screen objects also appear in **bold**.

*Italic* Variables and values that you must provide appear in *italics*. Words and phrases that are emphasized also appear in *italics*.

**Bold Italic** New terms appear in **bold italics** when they are defined in text.
Online Information

Click on the Read Me icon in the NetView Program Group for more information about the NetView program.

The online help facility provides task and user interface information.

The online books are available in HTML and PDF versions (DynaText is no longer supported). The HTML versions are accessible from the NetView Console using the Help..Books OnLine menu item, which will bring up the books in your default web browser.

PDF versions are available in the \usr\ov\books\c\pdf directory. If you have installed a non-English version of NetView, replace the c subdirectory with the appropriate locale specifier.

In addition, you can access online documents at this Web address:
http://www.tivoli.com/support/

A user name and password are required.

Accessability Information


Keyboard Access

Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

Refer to Tivoli NetView for Windows User's Guide for more information about keyboard access.

Contacting Tivoli Support

If you have a problem with any Tivoli product, you can contact Tivoli Customer Support. See the Tivoli Customer Support Handbook at the following Web address:

http://www.tivoli.com/support/handbook/

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:

• Registration and eligibility
• Telephone numbers and e-mail addresses, depending on the country you are in
• What information you should gather before contacting support
Chapter 1. Function Tables for NetView Reference 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

These tables describe the following functions:
- Graphical User Interface Routines
- SNMP Commands
- SNMP Routines
- Open Topology Functions
- GTM API Routines
- IP Topology Functions
- Client/Server APIs
- Miscellaneous Functions

<table>
<thead>
<tr>
<th>Graphical User Interface Routines and Reference Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine Name</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>OVwAckMapClose</td>
</tr>
<tr>
<td>OVwAckUserSubmapCreate</td>
</tr>
<tr>
<td>OVwAddActionCallback</td>
</tr>
<tr>
<td>OVwAddAlertCallback</td>
</tr>
<tr>
<td>OVwAddCallback</td>
</tr>
<tr>
<td>OVwAddHelpCallback</td>
</tr>
<tr>
<td>OVwAddInput</td>
</tr>
<tr>
<td>OVwAddMenuItem</td>
</tr>
<tr>
<td>OVwAddMenuItemFunction</td>
</tr>
<tr>
<td>OVwAlertMsg</td>
</tr>
<tr>
<td>OVwApiIntro</td>
</tr>
<tr>
<td>OVwBeginMapSync</td>
</tr>
<tr>
<td>OVwCheckAction</td>
</tr>
<tr>
<td>OVwClearBackgroundGraphic</td>
</tr>
<tr>
<td>OVwClearSymbolApp</td>
</tr>
<tr>
<td>OVwConfirmAddSymbolCB</td>
</tr>
<tr>
<td>OVwConfirmAppConfigCB</td>
</tr>
<tr>
<td>OVwConfirmCapabilityChangeCB</td>
</tr>
<tr>
<td>OVwConfirmCompoundStatusCB</td>
</tr>
<tr>
<td>Routine Name</td>
</tr>
<tr>
<td>--------------------------------------</td>
</tr>
<tr>
<td>OVwConfirmConnectSymbolsCB¹</td>
</tr>
<tr>
<td>OVwConfirmCreateObjectsCB¹</td>
</tr>
<tr>
<td>OVwConfirmCreateSubmapsCB¹</td>
</tr>
<tr>
<td>OVwConfirmCreateSymbolsCB¹</td>
</tr>
<tr>
<td>OVwConfirmDeleteObjectsCB¹</td>
</tr>
<tr>
<td>OVwConfirmDeleteSubmapsCB¹</td>
</tr>
<tr>
<td>OVwConfirmDeleteSymbolsCB¹</td>
</tr>
<tr>
<td>OVwConfirmDescribeCB¹</td>
</tr>
<tr>
<td>OVwConfirmHideSymbolsCB¹</td>
</tr>
<tr>
<td>OVwConfirmManageObjectsCB¹</td>
</tr>
<tr>
<td>OVwConfirmMoveSymbolCB¹</td>
</tr>
<tr>
<td>OVwConfirmObjectStatusCB¹</td>
</tr>
<tr>
<td>OVwConfirmSymbolStatusCB¹</td>
</tr>
<tr>
<td>OVwConfirmUnhideSymbolsCB¹</td>
</tr>
<tr>
<td>OVwConfirmUnmanageObjectsCB¹</td>
</tr>
<tr>
<td>OVwCopyMapInfo</td>
</tr>
<tr>
<td>OVwCreateAction</td>
</tr>
<tr>
<td>OVwCreateApp</td>
</tr>
<tr>
<td>OVwCreateComponentSymbol</td>
</tr>
<tr>
<td>OVwCreateComponentSymbolByName</td>
</tr>
<tr>
<td>OVwCreateConnSymbol</td>
</tr>
<tr>
<td>OVwCreateConnSymbolByName</td>
</tr>
<tr>
<td>OVwCreateMenu</td>
</tr>
</tbody>
</table>
### Table 1. Graphical User Interface Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVwCreateMenuItem</td>
<td>Creates a menu item in the current registration context</td>
<td>446</td>
</tr>
<tr>
<td>OVwCreateSubmap</td>
<td>Creates a submap</td>
<td>449</td>
</tr>
<tr>
<td>OVwCreateSymbol</td>
<td>Creates a symbol</td>
<td>453</td>
</tr>
<tr>
<td>OVwCreateSymbolByName</td>
<td>Creates a symbol representing the object with the name field value indicated by the specified name</td>
<td>453</td>
</tr>
<tr>
<td>OVwCreateSymbolByHostName</td>
<td>Creates a symbol representing the object identified by the specified IP host name</td>
<td>453</td>
</tr>
<tr>
<td>OVwCreateSymbolBySelectionName</td>
<td>Creates a symbol representing the object identified by the specified selection name</td>
<td>453</td>
</tr>
<tr>
<td>OVwCreateSymbols</td>
<td>Creates symbols</td>
<td>453</td>
</tr>
<tr>
<td>OVwDbAppendEnumConstants</td>
<td>Appends constants to an existing enumeration</td>
<td>466</td>
</tr>
<tr>
<td>OVwDbCreateField</td>
<td>Creates a field in the graphical user interface object database</td>
<td>468</td>
</tr>
<tr>
<td>OVwDbCreateObject</td>
<td>Creates an object in the graphical user interface object database</td>
<td>471</td>
</tr>
<tr>
<td>OVwDbCreateObjectByHostName</td>
<td>Creates an object in the graphical user interface object database through a host name</td>
<td>471</td>
</tr>
<tr>
<td>OVwDbCreateObjectBySelectionName</td>
<td>Creates an object in the graphical user interface object database through a selection name</td>
<td>471</td>
</tr>
<tr>
<td>OVwDbDeleteField</td>
<td>Deletes a field in the graphical user interface object database</td>
<td>468</td>
</tr>
<tr>
<td>OVwDbDeleteObject</td>
<td>Deletes an object from the graphical user interface object database</td>
<td>473</td>
</tr>
<tr>
<td>OVwDbFieldIdToFieldName</td>
<td>Returns the name of the field that has the specified field ID</td>
<td>475</td>
</tr>
<tr>
<td>OVwDbFieldNameToFieldId</td>
<td>Returns the field ID of the field that has the specified field name</td>
<td>475</td>
</tr>
<tr>
<td>OVwDbFreeEnumConstants</td>
<td>Frees the memory allocated for an OVwEnumConstants structure</td>
<td>477</td>
</tr>
<tr>
<td>OVwDbFreeFieldBindList</td>
<td>Frees the memory allocated for an OVwFieldBindList structure</td>
<td>486</td>
</tr>
<tr>
<td>OVwDbFreeFieldInfo</td>
<td>Frees the memory allocated for an OVwFieldInfo structure</td>
<td>480</td>
</tr>
<tr>
<td>OVwDbFreeFieldList</td>
<td>Frees the memory allocated for an OVwFieldList structure</td>
<td>488</td>
</tr>
<tr>
<td>OVwDbFreeFieldValue</td>
<td>Frees the memory allocated for an OVwFieldValue structure</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbFreeObjectFieldList</td>
<td>Frees the memory allocated for an OVwObjectFieldList structure</td>
<td>488</td>
</tr>
<tr>
<td>OVwDbFreeObjectIdList</td>
<td>Frees the memory allocated for an OVwObjectIdList structure</td>
<td>496</td>
</tr>
<tr>
<td>OVwDbGetCapabilityFieldValues</td>
<td>Returns a list of all field values for capability fields for a specified object</td>
<td>488</td>
</tr>
<tr>
<td>OVwDbGetEnumConstants</td>
<td>Returns a list of all text value sets in the enumerated data type</td>
<td>477</td>
</tr>
</tbody>
</table>
Table 1. Graphical User Interface Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVwDbGetEnumName</td>
<td>Translates an index into an enumerated constant</td>
<td>477</td>
</tr>
<tr>
<td>OVwDbGetEnumValue</td>
<td>Translates an enumerated constant into an index value</td>
<td>477</td>
</tr>
<tr>
<td>OVwDbGetFieldBooleanValue</td>
<td>Returns the Boolean value set for a field of type ovwBooleanField for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldEnumByName</td>
<td>Returns the value set for a field of type ovwEnumField for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldEnumByValue</td>
<td>Returns the value set for a field of type ovwEnumField for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldInfo</td>
<td>Returns information about an object database field</td>
<td>480</td>
</tr>
<tr>
<td>OVwDbGetFieldIntegerValue</td>
<td>Returns the integer value set for a field of type ovwInitField for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldStringValue</td>
<td>Returns the string value set for a field of type ovwStringField for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldValue</td>
<td>Returns the value of a specified field for a specified object</td>
<td>482</td>
</tr>
<tr>
<td>OVwDbGetFieldValues</td>
<td>Returns a list of all the field values for a specified object</td>
<td>486</td>
</tr>
<tr>
<td>OVwDbGetFieldValuesByObjects</td>
<td>Returns the value set for a specified field for a list of objects</td>
<td>488</td>
</tr>
<tr>
<td>OVwDbGetNameFieldValues</td>
<td>Returns a list of all field values for name fields for a specified object</td>
<td>488</td>
</tr>
<tr>
<td>OVwDbGetUniqObjectName</td>
<td>Returns a unique name for an object</td>
<td>490</td>
</tr>
<tr>
<td>OVwDbHostnameToObjectId</td>
<td>Returns the object ID for the object with the specified IP host name</td>
<td>492</td>
</tr>
<tr>
<td>OVwDbInit</td>
<td>Initializes the graphical user interface object database.</td>
<td>494</td>
</tr>
<tr>
<td>OVwDbListFields</td>
<td>Returns a list of object database fields</td>
<td>496</td>
</tr>
<tr>
<td>OVwDbListObjectsByFieldValue</td>
<td>Returns a list of objects, from the graphical user interface, object database that have a single specific value set for a field</td>
<td>499</td>
</tr>
<tr>
<td>OVwDbListObjectsByFieldValues</td>
<td>Returns a list of objects from the graphical user interface object database that have all the field values specified by a list of fields</td>
<td>499</td>
</tr>
<tr>
<td>OVwDbNameToObjectId</td>
<td>Returns the object ID of the object that has a specified name in a specified name field</td>
<td>502</td>
</tr>
<tr>
<td>OVwDbObjectIdToHostname</td>
<td>Returns the IP hostname for the object with the specified object ID</td>
<td>502</td>
</tr>
<tr>
<td>OVwDbObjectIdToSelectionName</td>
<td>Returns the selection name for the object with the specified object ID</td>
<td>504</td>
</tr>
<tr>
<td>OVwDbSelectionNameToObjectId</td>
<td>Returns the ObjectId of the object that has a selection name matching the value provided by selectionName</td>
<td>504</td>
</tr>
<tr>
<td>OVwDbSetEnumConstants</td>
<td>Sets values for a field of type ovwEnumField</td>
<td>506</td>
</tr>
<tr>
<td>OVwDbSetFieldBooleanValue</td>
<td>Sets values for a field of type ovwBooleanField</td>
<td>506</td>
</tr>
</tbody>
</table>
Table 1. Graphical User Interface Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVwDbSetFieldEnumByName</td>
<td>Sets by index the value of a field of type ovwEnumField</td>
<td>508</td>
</tr>
<tr>
<td>OVwDbSetFieldEnumByValue</td>
<td>Sets by value the value of a field of type ovwEnumField</td>
<td>508</td>
</tr>
<tr>
<td>OVwDbSetFieldIntegerValue</td>
<td>Sets the value for a field of type ovwIntegerField</td>
<td>508</td>
</tr>
<tr>
<td>OVwDbSetFieldStringValue</td>
<td>Sets a value for a field of type ovwStringField</td>
<td>508</td>
</tr>
<tr>
<td>OVwDbSetFieldValue</td>
<td>Sets the value for the field specified by fieldId for a specified object</td>
<td>508</td>
</tr>
<tr>
<td>OVwDbSetHostname</td>
<td>Sets the value of the IP Hostname field for a specified object</td>
<td>511</td>
</tr>
<tr>
<td>OVwDbSetSelectionName</td>
<td>Sets the value of the Selection Name field for a specified object</td>
<td>511</td>
</tr>
<tr>
<td>OVwDbUnsetFieldValue</td>
<td>Unsets the value of a specified field for a specified object</td>
<td>514</td>
</tr>
<tr>
<td>OVwDbUnsetFieldValues</td>
<td>Unsets the values of fields in a list for a specified object</td>
<td>516</td>
</tr>
<tr>
<td>OVwDeleteAction</td>
<td>Deletes the specified action in the current registration context</td>
<td>433</td>
</tr>
<tr>
<td>OVwDeleteApp</td>
<td>Deletes the specified NetView application registration</td>
<td>440</td>
</tr>
<tr>
<td>OVwDeleteMenu</td>
<td>Deletes the specified menu from the current registration context</td>
<td>444</td>
</tr>
<tr>
<td>OVwDeleteMenuItem</td>
<td>Deletes the specified menu item from the current registration context</td>
<td>446</td>
</tr>
<tr>
<td>OVwDeleteSubmap</td>
<td>Deletes a submap from the open map</td>
<td>448</td>
</tr>
<tr>
<td>OVwDeleteSymbol</td>
<td>Deletes the symbol identified by symbolId from the open map</td>
<td>453</td>
</tr>
<tr>
<td>OVwDeleteSymbols</td>
<td>Deletes symbols in a list from the open map</td>
<td>453</td>
</tr>
<tr>
<td>OVwDisplaySubmap</td>
<td>Displays a submap of the open map</td>
<td>515</td>
</tr>
<tr>
<td>OVwDoAction</td>
<td>Starts any registered application action</td>
<td>512</td>
</tr>
<tr>
<td>OVwDone</td>
<td>Terminates an application’s connection to the NetView program</td>
<td>517</td>
</tr>
<tr>
<td>OVwEndMapSync</td>
<td>Ends map synchronization phase</td>
<td>510</td>
</tr>
<tr>
<td>OVwEndSessionCB(^1)</td>
<td>Functions as a callback for an end-of-session event</td>
<td>518</td>
</tr>
<tr>
<td>OVwError</td>
<td>Specifies the error code set by the last OVw API call</td>
<td>520</td>
</tr>
<tr>
<td>OVwErrorMsg</td>
<td>Describes graphical user interface object database API error codes</td>
<td>521</td>
</tr>
<tr>
<td>OVwEventIntro</td>
<td>Introduces NetView graphical user interface events</td>
<td>522</td>
</tr>
<tr>
<td>OVwFileDescriptor</td>
<td>Accesses the NetView program’s communications channel</td>
<td>524</td>
</tr>
<tr>
<td>OVwFindMenuItem</td>
<td>Finds a menu item</td>
<td>525</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>OVwFreeActionRegInfo</td>
<td>Frees the memory allocated for an OVwActionRegInfo structure</td>
<td>436</td>
</tr>
<tr>
<td>OVwFreeAppRegInfo</td>
<td>Frees the memory allocated for an OVwAppRegInfo structure</td>
<td>440</td>
</tr>
<tr>
<td>OVwFreeMapInfo</td>
<td>Frees the memory allocated for an OVwMapInfo structure</td>
<td>544</td>
</tr>
<tr>
<td>OVwFreeObjectInfo</td>
<td>Frees the memory allocated for an OVwObjectInfo structure</td>
<td>550</td>
</tr>
<tr>
<td>OVwFreeObjectList</td>
<td>Frees the memory allocated for an OVwObjectList structure</td>
<td>569</td>
</tr>
<tr>
<td>OVwFreeSubmapInfo</td>
<td>Frees memory allocated for an OVwSubmapInfo structure</td>
<td>556</td>
</tr>
<tr>
<td>OVwFreeSubmapList</td>
<td>Frees the memory allocated for an OVwSubmapList structure</td>
<td>571</td>
</tr>
<tr>
<td>OVwFreeSymbolInfo</td>
<td>Frees memory allocated for an OVwSymbolInfo structure</td>
<td>558</td>
</tr>
<tr>
<td>OVwFreeSymbolList</td>
<td>Frees the memory allocated for an OVwSymbolList structure</td>
<td>572</td>
</tr>
<tr>
<td>OVwFreeSymbolTypeList</td>
<td>Frees the memory allocated for an OVwSymbolTypeList structure</td>
<td>577</td>
</tr>
<tr>
<td>OVwGetAction</td>
<td>Retrieves registration information for the specified action in the current registration context</td>
<td>436</td>
</tr>
<tr>
<td>OVwGetApp</td>
<td>Retrieves registration information for the application that is the current registration context</td>
<td>440</td>
</tr>
<tr>
<td>OVwGetAppConfigValues</td>
<td>Gets application configuration parameters</td>
<td>507</td>
</tr>
<tr>
<td>OVwGetClientList</td>
<td>Returns a list of clients attached to the server</td>
<td>531</td>
</tr>
<tr>
<td>OVwGetAppName</td>
<td>Returns the name of the running application</td>
<td>530</td>
</tr>
<tr>
<td>OVwGetConnSymbol</td>
<td>Gets a connection symbol</td>
<td>532</td>
</tr>
<tr>
<td>OVwGetFirstAction</td>
<td>Returns the name of the first action registered in the current registration context</td>
<td>535</td>
</tr>
<tr>
<td>OVwGetFirstMenuItem</td>
<td>Returns the ID of the first menu item registered in the current registration context</td>
<td>537</td>
</tr>
<tr>
<td>OVwGetFirstMenuItemFunction</td>
<td>Returns the type and argument for the first function bound to a menu item in the current registration context</td>
<td>540</td>
</tr>
<tr>
<td>OVwGetFirstRegContext</td>
<td>Returns the name of the first application in the NetView program’s list of registered applications</td>
<td>532</td>
</tr>
<tr>
<td>OVwGetMapInfo</td>
<td>Returns information about the open map</td>
<td>544</td>
</tr>
<tr>
<td>OVwGetMenuItem</td>
<td>Retrieves registration information for the specified menu item in the current registration context</td>
<td>446</td>
</tr>
<tr>
<td>OVwGetMenuItemMenu</td>
<td>Returns the ID of the menu to which an item is attached</td>
<td>546</td>
</tr>
<tr>
<td>OVwGetMenuItemPath</td>
<td>Returns the path location of a menu item in the NetView menu bar structure</td>
<td>536</td>
</tr>
<tr>
<td>OVwGetMenuPathSeparator</td>
<td>Returns the current string used to separate menu labels in a menu path string</td>
<td>546</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>OVwGetNextAction</td>
<td>Returns the name of the first action registered in the current registration context</td>
<td>535</td>
</tr>
<tr>
<td>OVwGetNextMenuItem</td>
<td>Returns the ID of the next menu item registered in the current registration context</td>
<td>537</td>
</tr>
<tr>
<td>OVwGetNextMenuItemFunction</td>
<td>Returns the type and argument for the next function bound to a menu item in the current registration context</td>
<td>540</td>
</tr>
<tr>
<td>OVwGetNextRegContext</td>
<td>Returns the name of the next application in the NetView program's list of registered applications</td>
<td>542</td>
</tr>
<tr>
<td>OVwGetObjectInfo</td>
<td>Returns map-specific object information</td>
<td>550</td>
</tr>
<tr>
<td>OVwGetRegContext</td>
<td>Returns the name of the current registration context</td>
<td>552</td>
</tr>
<tr>
<td>OVwGetSelections</td>
<td>Returns the list of selected objects on the open map</td>
<td>554</td>
</tr>
<tr>
<td>OVwGetSubmapInfo</td>
<td>Returns information about a submap on the open map</td>
<td>556</td>
</tr>
<tr>
<td>OVwGetSymbolInfo</td>
<td>Returns information about a symbol on the open map</td>
<td>558</td>
</tr>
<tr>
<td>OVwGetSymbolsByObject</td>
<td>Returns a list of all the symbols that represent an object on the open map</td>
<td>560</td>
</tr>
<tr>
<td>OVwHighlightObject</td>
<td>Highlights all the symbols representing a specified object on the open map</td>
<td>562</td>
</tr>
<tr>
<td>OVwHighlightObjects</td>
<td>Highlights all the symbols representing the objects in a list on the open map</td>
<td>562</td>
</tr>
<tr>
<td>OVwInit</td>
<td>Initializes an application's connection to the NetView program</td>
<td>564</td>
</tr>
<tr>
<td>OVwIsIdEqual</td>
<td>Compares graphical user interface API IDs</td>
<td>567</td>
</tr>
<tr>
<td>OVwIsIdNull</td>
<td>Tests a graphical user interface API ID to determine whether it is NULL</td>
<td>567</td>
</tr>
<tr>
<td>OVwListObjectsOnMap</td>
<td>Returns a filtered list of the objects on the open map</td>
<td>568</td>
</tr>
<tr>
<td>OVwListSubmaps</td>
<td>Returns a filtered list of the submaps on the open map</td>
<td>571</td>
</tr>
<tr>
<td>OVwListSymbols</td>
<td>Returns a filtered list of symbols on a submap of the open map</td>
<td>574</td>
</tr>
<tr>
<td>OVwListSymbolTypeCaps</td>
<td>Returns a list of the capability field values that would be set if you add an object to the map using this symbol type</td>
<td>577</td>
</tr>
<tr>
<td>OVwListSymbolTypes</td>
<td>Returns a list of all the currently registered symbol types</td>
<td>577</td>
</tr>
<tr>
<td>OVwLockRegUpdates</td>
<td>Acquires permission for the application to make subsequent calls that modify NetView registration information</td>
<td>579</td>
</tr>
<tr>
<td>OVwMainLoop</td>
<td>Defines a while loop that continuously processes NetView events and application-registered input events</td>
<td>581</td>
</tr>
<tr>
<td>OVwMapCloseCB</td>
<td>Functions as a callback for a map close event</td>
<td>583</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>OVwMapOpenCB¹</td>
<td>Functions as a callback for a map open event</td>
<td>555</td>
</tr>
<tr>
<td>OVwPeekInputEvent</td>
<td>Determines whether an application’s registered input source has input awaiting processing</td>
<td>557</td>
</tr>
<tr>
<td>OVwPeekOVwEvent</td>
<td>Determines whether a specific type of NetView event is awaiting processing</td>
<td>557</td>
</tr>
<tr>
<td>OVwPending</td>
<td>Determines whether an NetView event or application-registered event is awaiting processing</td>
<td>559</td>
</tr>
<tr>
<td>OVwProcessEvent</td>
<td>Processes a pending NetView event</td>
<td>560</td>
</tr>
<tr>
<td>OVwQueryAddSymbolCB¹</td>
<td>Functions as a callback for a query add symbol event</td>
<td>652</td>
</tr>
<tr>
<td>OVwQueryAppConfigCB¹</td>
<td>Functions as a callback for a query application configuration event</td>
<td>658</td>
</tr>
<tr>
<td>OVwQueryConnectSymbolsCB¹</td>
<td>Functions as a callback for a query connect symbols event</td>
<td>661</td>
</tr>
<tr>
<td>OVwQueryDeleteSymbolsCB¹</td>
<td>Functions as a callback for a query delete symbols event</td>
<td>665</td>
</tr>
<tr>
<td>OVwQueryDescribeCB¹</td>
<td>Functions as a callback for a query describe symbols event</td>
<td>668</td>
</tr>
<tr>
<td>OVwRegIntro</td>
<td>Introduces NetView graphical user interface registration files</td>
<td>591</td>
</tr>
<tr>
<td>OVwRenameRegContext</td>
<td>Changes the name of an NetView registration context</td>
<td>511</td>
</tr>
<tr>
<td>OVwRemoveActionCallback</td>
<td>Unregisters a callback for a registered action</td>
<td>377</td>
</tr>
<tr>
<td>OVwRemoveAlertCallback</td>
<td>Unregisters a callback for an NetView alert</td>
<td>379</td>
</tr>
<tr>
<td>OVwRemoveCallback</td>
<td>Unregisters procedures to process NetView events</td>
<td>382</td>
</tr>
<tr>
<td>OVwRemoveHelpCallback</td>
<td>Unregisters a callback for application help requests</td>
<td>384</td>
</tr>
<tr>
<td>OVwRemoveInput</td>
<td>Removes an event source</td>
<td>386</td>
</tr>
<tr>
<td>OVwRemoveMenuItem</td>
<td>Removes a menu item from a menu in current context</td>
<td>388</td>
</tr>
<tr>
<td>OVwRemoveMenuItemFunction</td>
<td>Removes a menu item function from a menu item in current context</td>
<td>390</td>
</tr>
<tr>
<td>OVwSaveRegUpdates</td>
<td>Saves modifications to registration information</td>
<td>517</td>
</tr>
<tr>
<td>OVw SelectListChangeCB¹</td>
<td>Functions as a callback for a selection list change event</td>
<td>516</td>
</tr>
<tr>
<td>OVwSetAction</td>
<td>Modifies registration information for the specified action in the current registration context</td>
<td>486</td>
</tr>
<tr>
<td>OVwSetApp</td>
<td>Modifies registration information for the application that is the current registration context.</td>
<td>486</td>
</tr>
<tr>
<td>OVwSetAppConfigValues</td>
<td>Sets application configuration parameters</td>
<td>527</td>
</tr>
<tr>
<td>OVwSetBackgroundGraphic</td>
<td>Sets the background graphic for a specified submap</td>
<td>624</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>OVwSetMenuItem</td>
<td>Modifies registration information for the specified menu item in the current registration context</td>
<td>446</td>
</tr>
<tr>
<td>OVwSetMenuPathSeparator</td>
<td>Sets the character string used to separate menu labels in a menu path string</td>
<td>538</td>
</tr>
<tr>
<td>OVwSetRegContext</td>
<td>Sets the name of the current registration context to that of the specified application</td>
<td>552</td>
</tr>
<tr>
<td>OVwSetStatusOnObject</td>
<td>Sets the status of all symbols on the open map of the specified object that have the symbol status source ovwObjectStatusSource</td>
<td>523</td>
</tr>
<tr>
<td>OVwSetStatusOnObjects</td>
<td>Sets the object status on multiple objects</td>
<td>623</td>
</tr>
<tr>
<td>OVwSetStatusOnSymbol</td>
<td>Sets the status of a specified symbol if the symbol has status source ovwSymbolStatusSource and if the application has permission to modify the symbol</td>
<td>523</td>
</tr>
<tr>
<td>OVwSetStatusOnSymbols</td>
<td>Sets the symbol status on symbols in a list</td>
<td>626</td>
</tr>
<tr>
<td>OVwSetSubmapName</td>
<td>Sets the name of a submap</td>
<td>526</td>
</tr>
<tr>
<td>OVwSetSymbolApp</td>
<td>Sets application interest in a symbol</td>
<td>528</td>
</tr>
<tr>
<td>OVwSetSymbolBehavior</td>
<td>Sets the behavior of a symbol</td>
<td>530</td>
</tr>
<tr>
<td>OVwSetSymbolLabel</td>
<td>Sets the label of a symbol</td>
<td>535</td>
</tr>
<tr>
<td>OVwSetSymbolPosition</td>
<td>Sets the position of a symbol</td>
<td>535</td>
</tr>
<tr>
<td>OVwSetSymbolStatusSource</td>
<td>Sets the status source of a symbol</td>
<td>540</td>
</tr>
<tr>
<td>OVwSetSymbolType</td>
<td>Sets the symbol type of a symbol</td>
<td>543</td>
</tr>
<tr>
<td>OVwShowHelp</td>
<td>Requests presentation of help information</td>
<td>626</td>
</tr>
<tr>
<td>OVwSubmapCloseCB¹</td>
<td>Functions as a callback for a submap close event</td>
<td>548</td>
</tr>
<tr>
<td>OVwSubmapOpenCB¹</td>
<td>Functions as a callback for a submap open event</td>
<td>548</td>
</tr>
<tr>
<td>OVwUndoRegUpdates</td>
<td>Destroys all changes to registration information since the last call to OVwSaveRegUpdates or OVwLockRegUpdates</td>
<td>517</td>
</tr>
<tr>
<td>OVwUnLockRegUpdates</td>
<td>Releases previously acquired update permissions</td>
<td>575</td>
</tr>
<tr>
<td>OVwUserSubmapCreateCB¹</td>
<td>Functions as a callback for a user submap create event</td>
<td>550</td>
</tr>
<tr>
<td>OVwVerifyAdd</td>
<td>Validates the addition of a symbol</td>
<td>552</td>
</tr>
<tr>
<td>OVwVerifyAppConfigChange</td>
<td>Validates a change of application configuration values</td>
<td>555</td>
</tr>
<tr>
<td>OVwVerifyConnect</td>
<td>Validates the user-selected connect operation for two symbols</td>
<td>551</td>
</tr>
<tr>
<td>OVwVerifyDeleteSymbol</td>
<td>Validates the deletion of symbols</td>
<td>555</td>
</tr>
<tr>
<td>OVwVerifyDescribeChange</td>
<td>Validates the change of information describing an object</td>
<td>558</td>
</tr>
<tr>
<td>polld</td>
<td>Monitors the state of other daemons and restarts them as needed.</td>
<td>571</td>
</tr>
<tr>
<td>QuickTest</td>
<td>Enhanced ping tool that will poll all managed interfaces for a node.</td>
<td>572</td>
</tr>
</tbody>
</table>
Table 1. Graphical User Interface Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>¹A callback for an event might not be generated if the event resulted from an API call instead of a graphical user interface (GUI) action.</td>
<td></td>
</tr>
</tbody>
</table>

### SNMP Commands

Table 2. SNMP Commands and Reference Pages

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>appmon</td>
<td>Functions as a NetView terminal output encapsulation tool</td>
<td>25</td>
</tr>
<tr>
<td>event</td>
<td>Sends the event to the trapd daemon</td>
<td>28</td>
</tr>
<tr>
<td>findroute</td>
<td>Determines IP routing using SNMP requests</td>
<td>31</td>
</tr>
<tr>
<td>graph</td>
<td>Graphs data collected by snmpCollect, live SNMP data, and user-supplied data</td>
<td>32</td>
</tr>
<tr>
<td>loadhosts</td>
<td>Loads IP topology database from %windir%\system32\drivers\etc\hosts</td>
<td>44</td>
</tr>
<tr>
<td>loadmib</td>
<td>Loads and unloads SNMP MIBs</td>
<td>46</td>
</tr>
<tr>
<td>nmdemandpoll</td>
<td>Polls a node for information</td>
<td>50</td>
</tr>
<tr>
<td>nmpolling</td>
<td>Provides topology and status polling configurations</td>
<td>70</td>
</tr>
<tr>
<td>nvsmmptrap</td>
<td>Issues an SNMP trap</td>
<td>270</td>
</tr>
<tr>
<td>rnetstat</td>
<td>Lists network information of a remote SNMP node</td>
<td>573</td>
</tr>
<tr>
<td>snmpCollect</td>
<td>Collects, compares, and stores SNMP MIB values</td>
<td>680</td>
</tr>
<tr>
<td>snmpget</td>
<td>Queries a node using SNMP Get Request</td>
<td>683</td>
</tr>
<tr>
<td>snmpnext</td>
<td>Queries a node using SNMP GetNext Request</td>
<td>685</td>
</tr>
<tr>
<td>snmpodump</td>
<td>Formats data collected by snmpCollect</td>
<td>687</td>
</tr>
<tr>
<td>snmpset</td>
<td>Issues an SNMP Set Request</td>
<td>690</td>
</tr>
<tr>
<td>snmpwalk</td>
<td>Uses the SNMP GetNext Request to query node for information</td>
<td>692</td>
</tr>
<tr>
<td>spmsur.exe</td>
<td>Acts as a wrapper to the webserver and mibserver Java™ applications, allowing ovsmpmd to exercise control over them.</td>
<td>694</td>
</tr>
</tbody>
</table>

### SNMP Routines

Table 3. SNMP Routines and Reference Pages

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvSnmpBlockingGetTable</td>
<td>Retrieves an entire table from the MIB in a blocking manner</td>
<td>266</td>
</tr>
<tr>
<td>nvSnmpErrString</td>
<td>Returns SNMP-specific error strings</td>
<td>269</td>
</tr>
<tr>
<td>Command Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>nvSnmpGetTable</td>
<td>Retrieves an entire table from the MIB in a nonblocking manner</td>
<td>266</td>
</tr>
<tr>
<td>nvSnmpGetTableElement</td>
<td>Retrieves the specified element from the OVsnpVarBind structure returned by nvSnmpBlockingGetTable or one of its related functions</td>
<td>266</td>
</tr>
<tr>
<td>nvSnmpWGetTable</td>
<td>Retrieves an entire table from the MIB if XtMainLoop or an equivalent function is used to manage file I/O multiplexing</td>
<td>266</td>
</tr>
<tr>
<td>OVFree</td>
<td>Frees the memory allocated by the NetView API</td>
<td>281</td>
</tr>
<tr>
<td>OVFreeSetupSelect</td>
<td>Frees the memory allocated by OVSetupSelect</td>
<td>282</td>
</tr>
<tr>
<td>OVSelect</td>
<td>Selects the socket where data is written or read</td>
<td>283</td>
</tr>
<tr>
<td>OVSnpPDUCommunityName</td>
<td>Sets the Community Name field of a PDU structure</td>
<td>284</td>
</tr>
<tr>
<td>OVSnpPDUIpAddress</td>
<td>Sets the IP address field of a PDU structure</td>
<td>295</td>
</tr>
<tr>
<td>OVSetupSelect</td>
<td>Builds a list of sockets in which data is written or read</td>
<td>296</td>
</tr>
<tr>
<td>OVSnpAddNullVarBind</td>
<td>Creates and initializes a new OVsnpVarBind data structure</td>
<td>297</td>
</tr>
<tr>
<td>OVSnpAddTypedVarBind</td>
<td>Creates and initializes a new OVsnpVarBind data structure and allocates space for the value of the variable</td>
<td>297</td>
</tr>
<tr>
<td>OVSnpAddVarBind</td>
<td>Allocates space for and initializes an OVsnpVarBind data structure for getting and setting variables</td>
<td>297</td>
</tr>
<tr>
<td>OVSnpBlockingSend</td>
<td>Sends an SNMP PDU and receives the response</td>
<td>299</td>
</tr>
<tr>
<td>OVSnpClose</td>
<td>Frees resources allocated by a session created by a call to OVSnpOpen</td>
<td>302</td>
</tr>
<tr>
<td>ovsnmp.conf</td>
<td>Describes NetView SNMP and status polling configuration</td>
<td>304</td>
</tr>
<tr>
<td>OVSnp.ConfAllocEntry</td>
<td>Allocates dynamic storage for an OVSnpConfEntry structure</td>
<td>307</td>
</tr>
<tr>
<td>OVSnp.ConfClose</td>
<td>Closes an SNMP Configuration Database</td>
<td>308</td>
</tr>
<tr>
<td>OVSnp.ConfCopyEntry</td>
<td>Allocates a new OVSnpConfEntry and copies the contents of the old OVSnpConfEntry to the new one</td>
<td>309</td>
</tr>
<tr>
<td>OVSnp.ConfDeleteEntry</td>
<td>Deletes a record from the SNMP Configuration Database</td>
<td>310</td>
</tr>
<tr>
<td>OVSnp.ConfFreeDest</td>
<td>Frees an OVSnpConfDest structure and its contents</td>
<td>311</td>
</tr>
<tr>
<td>OVSnp.ConfFreeEntry</td>
<td>Frees an OVSnpConfEntry structure and its contents</td>
<td>312</td>
</tr>
<tr>
<td>OVSnp.ConfOpen</td>
<td>Opens an SNMP Configuration database for subsequent use</td>
<td>313</td>
</tr>
<tr>
<td>OVSnp.ConfReadDefault</td>
<td>Reads the global default parameters in the SNMP Configuration Database</td>
<td>316</td>
</tr>
</tbody>
</table>
### Table 3. SNMP Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVsnmp.ConfReadEntry</td>
<td>Reads the parameters for the target node from the SNMP Configuration Database</td>
<td>317</td>
</tr>
<tr>
<td>OVsnmp.ConfResolveDest</td>
<td>Returns the resolved SNMP configuration parameters for a target mode</td>
<td>319</td>
</tr>
<tr>
<td>OVsnmp.ConfStoreDefault</td>
<td>Stores the global default SNMP configuration parameters in the SNMP Configuration Database</td>
<td>321</td>
</tr>
<tr>
<td>OVsnmp.ConfStoreEntry</td>
<td>Stores the SNMP Configuration parameters for a target in the SNMP Configuration Database</td>
<td>322</td>
</tr>
<tr>
<td>OVsnmpCreatePdu</td>
<td>Allocates an OVsnmpPdu data structure of the specified type</td>
<td>326</td>
</tr>
<tr>
<td>OVsnmpDoRetry</td>
<td>Retransmits pending SNMP requests</td>
<td>327</td>
</tr>
<tr>
<td>OVsnmpErrString</td>
<td>Returns SNMP-specific error strings</td>
<td>328</td>
</tr>
<tr>
<td>OVsnmpFixPdu</td>
<td>Deletes a variable with an error from an SNMP PDU</td>
<td>331</td>
</tr>
<tr>
<td>OVsnmpFreePdu</td>
<td>Frees all memory associated with the specified PDU</td>
<td>333</td>
</tr>
<tr>
<td>OVsnmpGetRetryInfo</td>
<td>Gets information about pending SNMP requests to be retransmitted</td>
<td>335</td>
</tr>
<tr>
<td>OVsnmpIntro</td>
<td>Introduces the ovsnpm library</td>
<td>337</td>
</tr>
<tr>
<td>OVsnmpOpen</td>
<td>Establishes an active SNMP session for communication with an SNMP agent</td>
<td>342</td>
</tr>
<tr>
<td>OVsnmpRead</td>
<td>Receives SNMP messages on all active sessions</td>
<td>345</td>
</tr>
<tr>
<td>OVsnmpRecv</td>
<td>Receives an SNMP PDU for a specified session</td>
<td>347</td>
</tr>
<tr>
<td>OVsnmpSend</td>
<td>Sends an SNMP PDU in non-blocking mode</td>
<td>349</td>
</tr>
<tr>
<td>OVsnmpTrapOpen</td>
<td>Connects to the trapd daemon and sets up to receive traps in a non-Windows environment</td>
<td>352</td>
</tr>
<tr>
<td>OVsnmpWClose</td>
<td>Frees resources allocated by a session created by a call to OVsnmpWOpen in a Windows environment</td>
<td>353</td>
</tr>
<tr>
<td>OVsnmpWOpen</td>
<td>Establishes an active SNMP session for communication with an SNMP agent in a Windows environment</td>
<td>342</td>
</tr>
<tr>
<td>OVsnmpWSend</td>
<td>Sends an SNMP PDU in non-blocking mode in a Windows environment</td>
<td>349</td>
</tr>
<tr>
<td>OVsnmpWTTrapOpen</td>
<td>Connects to the trapd daemon and sets up to receive traps in a Windows environment</td>
<td>352</td>
</tr>
</tbody>
</table>

### Open Topology Functions

**Table 4. Open Topology Functions and Reference Pages**

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid_to_protocol</td>
<td>Provides a table used to convert from an oid provided in the NetView topology MIB to a string that represents the graph protocol name</td>
<td>272</td>
</tr>
</tbody>
</table>
### Table 4. Open Topology Functions and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxmap</td>
<td>Provides the display of open topology information stored in gtmd</td>
<td>711</td>
</tr>
</tbody>
</table>

### GTM API Routines

Table 5. GTM API Routines and Reference Pages

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvotChangeArcDetails</td>
<td>Changes the contents of the <em>details</em> variable for the specified arc</td>
<td>81</td>
</tr>
<tr>
<td>nvotChangeArcIconInGraph</td>
<td>Changes the icon representing an arc in a graph</td>
<td>85</td>
</tr>
<tr>
<td>nvotChangeArcLabelInGraph</td>
<td>Changes the label on an arc in a graph</td>
<td>80</td>
</tr>
<tr>
<td>nvotChangeArcStatus</td>
<td>Changes one or more status values of an arc</td>
<td>95</td>
</tr>
<tr>
<td>nvotChangeBoxBackground</td>
<td>Changes the background image for the child submap of a box graph</td>
<td>98</td>
</tr>
<tr>
<td>nvotChangeBoxDetails</td>
<td>Changes the contents of the <em>details</em> variable for the specified box graph</td>
<td>101</td>
</tr>
<tr>
<td>nvotChangeBoxIconInGraph</td>
<td>Changes the icon representing a box graph in a graph</td>
<td>104</td>
</tr>
<tr>
<td>nvotChangeBoxLabelInGraph</td>
<td>Changes the label on a box in a graph</td>
<td>107</td>
</tr>
<tr>
<td>nvotChangeBoxPositionInGraph</td>
<td>Changes the position of a box graph icon in a graph submap</td>
<td>110</td>
</tr>
<tr>
<td>nvotChangeGraphBackground</td>
<td>Changes the background image for the child submap of a graph</td>
<td>116</td>
</tr>
<tr>
<td>nvotChangeGraphDetails</td>
<td>Changes the contents of the <em>details</em> variable for the specified graph</td>
<td>117</td>
</tr>
<tr>
<td>nvotChangeGraphIconInGraph</td>
<td>Changes the icon representing a graph in a graph</td>
<td>118</td>
</tr>
<tr>
<td>nvotChangeGraphLabelInGraph</td>
<td>Changes the label on a graph in a graph</td>
<td>121</td>
</tr>
<tr>
<td>nvotChangeGraphPositionInGraph</td>
<td>Changes the position of a graph icon in a graph submap</td>
<td>122</td>
</tr>
<tr>
<td>nvotChangeRootGraphIcon</td>
<td>Changes the icon representing the root graph</td>
<td>123</td>
</tr>
<tr>
<td>nvotChangeRootGraphLabel</td>
<td>Changes the label on the root graph</td>
<td>124</td>
</tr>
<tr>
<td>nvotChangeVertexDetails</td>
<td>Changes the contents of the <em>details</em> variable for the specified vertex</td>
<td>134</td>
</tr>
<tr>
<td>nvotChangeVertexIconInBox</td>
<td>Changes the icon representing a vertex in a box graph</td>
<td>137</td>
</tr>
<tr>
<td>nvotChangeVertexIconInGraph</td>
<td>Changes the icon representing a vertex in a graph</td>
<td>139</td>
</tr>
<tr>
<td>nvotChangeVertexLabelInBox</td>
<td>Changes the label on a vertex in a box graph</td>
<td>140</td>
</tr>
<tr>
<td>nvotChangeVertexLabelInGraph</td>
<td>Changes the label on a vertex in a graph</td>
<td>141</td>
</tr>
<tr>
<td>nvotChangeVertexPositionInBox</td>
<td>Changes the position of a vertex icon in a box graph submap</td>
<td>142</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>nvotChangeVertexPositionInGraph</td>
<td>Changes the position of a vertex icon in a graph submap</td>
<td>152</td>
</tr>
<tr>
<td>nvotChangeVertexStatus</td>
<td>Changes one or more status values of a vertex</td>
<td>155</td>
</tr>
<tr>
<td>nvotCreateArcInGraph</td>
<td>Creates an arc in a graph</td>
<td>158</td>
</tr>
<tr>
<td>nvotCreateBoxInGraph</td>
<td>Creates a box graph in a graph</td>
<td>164</td>
</tr>
<tr>
<td>nvotCreateGraphInGraph</td>
<td>Creates a graph in a graph</td>
<td>168</td>
</tr>
<tr>
<td>nvotCreateProvidingSap</td>
<td>Creates a SAP of type providing</td>
<td>174</td>
</tr>
<tr>
<td>nvotCreateRootGraph</td>
<td>Creates a root graph</td>
<td>178</td>
</tr>
<tr>
<td>nvotCreateUsingSap</td>
<td>Creates a SAP of type using</td>
<td>182</td>
</tr>
<tr>
<td>nvotCreateVertexInBox</td>
<td>Creates a vertex in a box graph</td>
<td>185</td>
</tr>
<tr>
<td>nvotCreateVertexInGraph</td>
<td>Creates a vertex in a graph</td>
<td>189</td>
</tr>
<tr>
<td>nvotDeleteArc</td>
<td>Deletes an arc</td>
<td>193</td>
</tr>
<tr>
<td>nvotDeleteArcFromGraph</td>
<td>Deletes an arc from a graph</td>
<td>196</td>
</tr>
<tr>
<td>nvotDeleteBox</td>
<td>Deletes a box graph</td>
<td>200</td>
</tr>
<tr>
<td>nvotDeleteBoxFromGraph</td>
<td>Deletes a box graph from a graph</td>
<td>203</td>
</tr>
<tr>
<td>nvotDeleteGraph</td>
<td>Deletes a graph</td>
<td>205</td>
</tr>
<tr>
<td>nvotDeleteGraphFromGraph</td>
<td>Deletes a graph from a graph</td>
<td>207</td>
</tr>
<tr>
<td>nvotDeleteProvidingSap</td>
<td>Deletes a SAP of type providing</td>
<td>210</td>
</tr>
<tr>
<td>nvotDeleteUsingSap</td>
<td>Deletes a SAP of type using</td>
<td>213</td>
</tr>
<tr>
<td>nvotDeleteVertex</td>
<td>Deletes a vertex</td>
<td>216</td>
</tr>
<tr>
<td>nvotDeleteVertexFromBox</td>
<td>Deletes a vertex from a box graph</td>
<td>218</td>
</tr>
<tr>
<td>nvotDeleteVertexFromGraph</td>
<td>Deletes a vertex from a graph</td>
<td>221</td>
</tr>
<tr>
<td>nvotDone</td>
<td>Closes the socket connection to gtmd</td>
<td>224</td>
</tr>
<tr>
<td>nvotFree</td>
<td>Frees memory allocated by a get routine</td>
<td>226</td>
</tr>
<tr>
<td>nvotGetArcsInGraph</td>
<td>Gets a list of all arcs contained in a graph</td>
<td>228</td>
</tr>
<tr>
<td>nvotGetBoxesInGraph</td>
<td>Gets a list of all box graphs contained in a graph</td>
<td>231</td>
</tr>
<tr>
<td>nvotGetError</td>
<td>Retrieves the error code set by the last function call</td>
<td>234</td>
</tr>
<tr>
<td>nvotGetErrorMsg</td>
<td>Converts a return code into a string</td>
<td>237</td>
</tr>
<tr>
<td>nvotGetGraphsInGraph</td>
<td>Gets a list of all graphs contained in a graph</td>
<td>238</td>
</tr>
<tr>
<td>nvotGetSapsOnVertex</td>
<td>Gets a list of all SAPs associated with a vertex</td>
<td>241</td>
</tr>
<tr>
<td>nvotGetVerticesInBox</td>
<td>Gets a list of all vertices contained in a box graph</td>
<td>244</td>
</tr>
<tr>
<td>nvotGetVerticesInGraph</td>
<td>Gets a list of all vertices contained in a graph</td>
<td>247</td>
</tr>
<tr>
<td>nvotInit</td>
<td>Opens a socket connection to gtmd</td>
<td>250</td>
</tr>
<tr>
<td>nvotSetCenterBoxForGraph</td>
<td>Specifies which box graph icon is to be the center of a star graph submap</td>
<td>253</td>
</tr>
<tr>
<td>nvotSetCenterGraphForGraph</td>
<td>Specifies which graph icon is to be the center of a star graph submap</td>
<td>255</td>
</tr>
<tr>
<td>nvotSetSynchronousCreation</td>
<td>Specifies whether OVw object IDs are to be returned in synchronous mode</td>
<td>256</td>
</tr>
</tbody>
</table>
Table 5. GTM API Routines and Reference Pages (continued)

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvplay</td>
<td>provides audio when a specific trap is received</td>
<td>261</td>
</tr>
<tr>
<td>nvservice</td>
<td>manages the NetView Service from the command line</td>
<td>262</td>
</tr>
<tr>
<td>nvsniffer</td>
<td>Discovers services on nodes in the network and monitors the status of these services</td>
<td>263</td>
</tr>
</tbody>
</table>

**IP Topology Functions**

Table 6. IP Topology Functions and Reference Pages

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipmap</td>
<td>Displays map of network discovered by netmon using the NetView graphical user interface</td>
<td>37</td>
</tr>
<tr>
<td>oid_to_sym</td>
<td>Maps SNMP sysObjectld to NetView symbol types</td>
<td>273</td>
</tr>
<tr>
<td>oid_to_type</td>
<td>Maps SNMP sysObjectld to vendor and type</td>
<td>275</td>
</tr>
<tr>
<td>ovtopodump</td>
<td>Prints the contents of the ovtopmd database</td>
<td>266</td>
</tr>
</tbody>
</table>

**Client/Server APIs**

Table 7. Client/Server APIs and Reference Pages

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVisClient</td>
<td>Checks to see if an application is running on a client or a server</td>
<td>80</td>
</tr>
<tr>
<td>NVgetServerName</td>
<td>Determines the name of the server to which a client should connect</td>
<td>78</td>
</tr>
<tr>
<td>NVinstallMode</td>
<td>Returns the installation mode of the NetView program: client, server, or single user mode.</td>
<td>79</td>
</tr>
</tbody>
</table>

**Miscellaneous Functions**

Table 8. Miscellaneous Functions and Reference Pages

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addtrap</td>
<td>Adds a trap definition to the trapd.conf file</td>
<td>20</td>
</tr>
<tr>
<td>lrf</td>
<td>Provides the format for NetView local registration files</td>
<td>48</td>
</tr>
<tr>
<td>mibExpr.conf</td>
<td>Associates a name with an SNMP MIB expression</td>
<td>51</td>
</tr>
<tr>
<td>mibform</td>
<td>Gets and displays MIB objects in simple form format</td>
<td>53</td>
</tr>
<tr>
<td>mibserver</td>
<td>Runs the MIB server.</td>
<td>55</td>
</tr>
<tr>
<td>mibtable</td>
<td>Gets and displays an arbitrary MIB table</td>
<td>56</td>
</tr>
<tr>
<td>Command Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>mib2trap</td>
<td>Automatically customizes trap definitions using a MIB definition file</td>
<td>57</td>
</tr>
<tr>
<td>netview</td>
<td>Displays the graphical user interface for the NetView program</td>
<td>58</td>
</tr>
<tr>
<td>netviewd</td>
<td>The netviewd daemon runs unattended and serves as a map server to the Web Clients.</td>
<td>68</td>
</tr>
<tr>
<td>nvcold</td>
<td>Maintains SmartSets as they have been defined by users or applications.</td>
<td>72</td>
</tr>
<tr>
<td>nvcoll</td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>nvecho</td>
<td>Provides a visual notification when an event occurs</td>
<td>74</td>
</tr>
<tr>
<td>nvmail</td>
<td>Sends a mail message</td>
<td>75</td>
</tr>
<tr>
<td>nvpage</td>
<td>Provides a way to submit a page to the system-wide pager request queue</td>
<td>76</td>
</tr>
<tr>
<td>ovaddobj</td>
<td>Registers the information necessary to start up object managers and information about the object managers and the objects they represent</td>
<td>277</td>
</tr>
<tr>
<td>ovdelobj</td>
<td>Functions as the NetView object deregistration utility</td>
<td>278</td>
</tr>
<tr>
<td>ovmapcount</td>
<td>Checks map reference counts</td>
<td>283</td>
</tr>
<tr>
<td>ovmapdump</td>
<td>Prints the contents of the graphical user interface map database</td>
<td>285</td>
</tr>
<tr>
<td>ovmapsnap</td>
<td>Creates a snapshot of a map</td>
<td>287</td>
</tr>
<tr>
<td>OVmib_get_objid_name</td>
<td>Converts a MIB variable object ID to its textual name</td>
<td>288</td>
</tr>
<tr>
<td>OVmib_read_objid</td>
<td>Converts a MIB variable name to its object ID format</td>
<td>289</td>
</tr>
<tr>
<td>ovobjprint</td>
<td>Displays the contents of the graphical user interface database</td>
<td>290</td>
</tr>
<tr>
<td>OVsDone</td>
<td>Notifies ovsppmd that a well-behaved object manager is exiting</td>
<td>356</td>
</tr>
<tr>
<td>OVsInit</td>
<td>Returns a file descriptor for interprocess communication with ovsppmd</td>
<td>356</td>
</tr>
<tr>
<td>OVsInitComplete</td>
<td>Notifies ovsppmd when a well-behaved object manager has finished initializing</td>
<td>356</td>
</tr>
<tr>
<td>OVsPMD_API</td>
<td>Describes routines for well-behaved daemon processes in the NetView program</td>
<td>356</td>
</tr>
<tr>
<td>OVsReceive</td>
<td>Receives a command from ovsppmd when called by a well-behaved object manager</td>
<td>356</td>
</tr>
<tr>
<td>ovstart</td>
<td>Starts NetView daemon processes</td>
<td>358</td>
</tr>
<tr>
<td>ovstatus</td>
<td>Reports the status of NetView daemon processes</td>
<td>360</td>
</tr>
<tr>
<td>ovstop</td>
<td>Stops NetView daemon processes</td>
<td>361</td>
</tr>
<tr>
<td>ovw_config</td>
<td>Compiles bitmaps and user-defined symbols</td>
<td>370</td>
</tr>
<tr>
<td>ovw_fields</td>
<td>Creates the fields defined in the currently installed registration files</td>
<td>371</td>
</tr>
<tr>
<td>Command Name</td>
<td>Description</td>
<td>Reference Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ovw_verify</td>
<td>Checks the syntax for all installed application, symbol type, and field registration files</td>
<td>372</td>
</tr>
<tr>
<td>thresh</td>
<td>Sets a threshold for MIB data collection</td>
<td>593</td>
</tr>
<tr>
<td>trapd</td>
<td>Receives, logs, and forwards events and traps</td>
<td>700</td>
</tr>
<tr>
<td>trapd.conf</td>
<td>Functions as the configuration file for trap and event formats</td>
<td>703</td>
</tr>
<tr>
<td>trapfrwd</td>
<td>Forwards traps to another management station</td>
<td>703</td>
</tr>
<tr>
<td>webserver</td>
<td>Runs the jetty server for the Tivoli NetView web console.</td>
<td>710</td>
</tr>
</tbody>
</table>
Chapter 2. Reference Pages

This chapter contains the reference pages for the NetView program. These reference pages are organized alphabetically.

Format of Reference Pages

The detailed descriptions of NetView commands, daemons, files, and applications follow a standard format. Each reference page can 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 a 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, Windows NT documentation, Internet Request for Comments, and other information sources
addtrap

Purpose

Adds a trap definition to the trapd.conf file.

Syntax

```
addtrap -n enterprise-name -l trap-label -i enterprise-object-id -g generic-trap -s specific-trap -S severity -o source-id -t status_type -c category [-C command -A command-argument] [-d display type]
addtrap @filename

addtrap -D -i enterprise-object-id -g generic-trap -s specific-trap
```

Description

The `addtrap` command creates a trap and adds the new trap to the `/usr/ov/conf/trapd.conf` file. If there is no enterprise definition for the trap, the new enterprise definition is added. If a trap exists with identical enterprise-object-id, generic-trap, and specific-trap values, the addtrap command updates the existing trap with the new information. After updating the `/usr/ov/conf/trapd.conf` file, the addtrap command sends an event to the trapd daemon informing it of the update.

Use the `-D` flag to remove trap definitions from the `/usr/ov/conf/trapd.conf` file.

The graphical user interface equivalent to the addtrap command is the Options..Events..Trap Settings.

Flags

- `-n enterprise-name`
  Specifies the enterprise name, a label used to identify the vendor of the enterprise. This should correspond to the enterprise name as defined by the enterprise-specific MIB. Type the enterprise name as a character string.

- `-l trap-label`
  Specifies the label identifying the trap. The default label is ADDTRAP-ADDED. Type the label as a character string.

- `-i enterprise-object-id`
  Specifies the enterprise object ID in valid, dot-notation format. For example, type: 1.3.6.1.4.1.2.6.3.

- `-g generic-trap`
  Specifies the generic trap number as an integer. The generic traps 0-through-5 are defined by SNMP and assume a specific-trap number of 0 (zero). The generic trap number 6 indicates an enterprise-specific trap.

- `-s specific-trap`
  Specifies the enterprise-specific trap number. This value is not required if the generic trap number is in the range of 0–5. If the generic trap number is 6, type a number for the specific trap in integer format.

- `-S severity`
  Specifies the severity of a trap, which can be one of the following:
  0     Cleared
1 Indeterminate
2 Warning
3 Minor
4 Critical
5 Major

-o source-id
Specifies the origin of a trap. Type one of the following characters to identify the
trap source. This character is in the Source column of the
\usr\etc\nm\conf\trapd.conf file and on the Event Cards and List.
A Agent
C Collect
d Demo
D Data Collector
E nvevents
I IPmap
L loadmib
M IP topology
n netmon related
N netmon-generated traps
P Open trap (other than IP)
S Security Agent
t NetView trap
T trapd
V Vendor-related traps
? Unknown

-t status-type
Specifies the status that is to be assigned to an object when it generates the
defined trap. The following single characters are valid:
0 Defaults
1 Unknown
2 Up
3 Marginal
4 Down
5 Unmanaged
6 Acknowledge
7 User1
8 User2
9 Unreachable

-c category
Specifies the event category under which the trap will be grouped in the Event
Cards and List. If you select “LOGONLY” this event is not displayed in the
Event Cards and List. Type one of the following values as a string of characters
enclosed in double quotation marks:
• “LOGONLY”
• “Threshold Events”
• “Network Topology Events”
• “Error Events”
• “Status Events”
• “Node Configuration Events”
• “Application Alert Events”

-f command-flag
Specifies whether a specified action will be performed by the management
system when the trap is received. Type one of the following values:
- A dash indicates that no action command is defined.
!

An exclamation mark indicates that an action command is defined. If you specify an exclamation mark, you must enter the -C and -A flags.

-F format-specification

Specifies the information that is entered in the Description column of the \usr\ov\log\trapd.log file and the Event Cards and List. Type a string of characters enclosed in single quotation marks.

You can enter several special characters to give control of the formatted output. The format can include standard C printf formats, and you can include information from the incoming trap by using the $<arg> format specification, with arg being one of the following:

$C Display the trap community string.
$E Display the enterprise as a text string, if possible; otherwise, display as in the $e arg.
$e Display the enterprise as an Object ID.
$A Display the trap agent-addr using gethostbyaddr as a string. If this fails, the program uses inettoa to print as an IP address.
$G Display the generic-trap.
$S Display the specific-trap.
$T Display the time stamp. This is the remote machine’s time, in hundredths of a second, between the last initialization of the device and the generation of the trap.
$* Display all the variables supplied by the trap as name-type-value strings.
$# Display the number of variables in the trap.
$$ Display the ‘$’ character.
$n Display the value of the n’th variable in the trap, where n is the variable sequence number starting at 1 as it appears in variable bindings.
$−n Display variable #n as a name-type:value string.
$+n Display variable #n as a name:value string.
$%n Display corresponding text associated with the codepoint defined in variable #n. If the next variable is associated with the subvector, print that text also. Continue until the next variable is not associated with the subvector.

-C command

Specifies the name of the command or program to be executed when the trap is received. This value is required when you enter a value of ! with the -f flag. Type a string of characters enclosed in double quotation marks.

There are two valid command types: the wecho command and operating system commands. The wecho command consists of the keyword, wecho, followed by a message. The message is displayed in a newly created error window. Operating system commands are executed by the event handling routines when the trap is received. Typical commands might include sending mail to a system administrator or invoking a program supplied by a vendor to manage the device.
-A command-argument
   Specifies the argument list that is passed to the command or program. This value is required when you enter a value of ! with the -f flag.
   Type a character string enclosed in double quotation marks. The special characters defined for the -F option are valid.

-d display type
   Indicates the display type of the application specified (if any) for the action command. Valid values are hidden, console, and windows.

@ filename
   Contains addtrap commands. The addtrap command reads the specified file and processes the commands. If you use this parameter, place it first in the addtrap command line. When you create an addtrap command file, it should contain:
   - One or more complete addtrap commands with all required flags.
   - Do not include tabs.
   - Carefully enter traps in the correct format. Incorrectly formatted traps cause the addtrap program to end and the trapd.conf file is not changed.

Because this parameter ignores other command line parameters, enter it alone. Using this parameter speeds execution of addtrap commands.

-D delete
   Remove the specified trap from the /usr/ov/conf/trapd.conf file. The -i, -g, and -s flags must also be specified to describe the enterprise ID, generic trap ID, and specific trap ID of the trap to be deleted.

Implementation Specifics

The addtrap command supports single-byte character code sets.

Examples

`/usr/ov/bin/addtrap -l newtrap -n ibm -i 9.9.9.9 -g 1 -o A -s 0 -t 1 -c "Status Events" -f - -F $$ -S 1`

or

`/usr/ov/bin/addtrap -l newtrap -n dec -i 9.9.9.9 -g -1 -o A -s 0 -t 1 -c "Status Events" -f ! -F $$ -S 1 -C "wecho Help" -A help`

The following example deletes the AppUp trap in the NetView enterprise.

`/usr/ov/bin/addtrap -D -i 1.3.6.1.4.1.2.6.3 -g 6 -s 59179056`

where

- -i 1.3.6.1.4.1.2.6.3 specifies the NetView enterprise
- -g 6 specifies the generic trap id used by the NetView traps
- -s 59179056 specifies the specific trap ID in the NetView enterprise which is to be deleted.

Files

- /usr/ov/conf/trapd.conf
   The trapd daemon configuration file

- /usr/ov/log/trapd.log
   The trapd daemon log file

Related Information

- See [trapd](#) on page 705.
- See [trapd.conf](#) on page 708.
• See `mib2trap(1)` on page 57.
appmon

Purpose

Starts the NetView Terminal Output Encapsulation Tool

Syntax

appmon [-options]

Description

The **appmon** command enables a developer to wrap an output-only terminal-based application to be used with the NetView graphical user interface. The appmon command sorts, prints, or saves the output to a file. It also gives a standard NetView appearance to applications.

Flags

The following options are available.

- **-cmd <shell_command>**
  This is the only required option. It is the command whose output is displayed inside the appmon window. The `<shell_command>` can include parameters. It is not enclosed in quotation marks.

- **-helpFile <filename>**
  The name of a file that contains the help information about the application. The full path must be specified for the help file and stored in the same directory as the netview.hlp file, \usr\ov\help\c.

  The help file for the appmon application you write must be in Windows NT help format. The help file is displayed when the user presses the Help button on the appmon window displaying your application.

  If no helpKey is specified, WinHelp is called with the command HELP_CONTENTS. See the *Programmer's Guide* for more information on integrating online help with NetView applications.

- **-helpKey <sequence number>**
  The sequence number of the help topic. Use this flag in conjunction with the -helpFile flag to specify a help topic from your help file. Enter the sequence number generated by the help authoring tool to specify the topic. WinHelp is called with the sequence number and the command HELP_CONTEXT. If you omit the -helpKey flag, the contents of the help file is used.

- **-printerCommand <command>**
  Command to which the data output is sent.

- **-saveFile <filename>**
  The name of an ASCII file to which the screen contents are dumped when the File..Save As menu item is selected.

- **-table**
  Displays information in a sortable table with a bold font. If you omit this flag, the information is displayed in a single column with a light monospaced font, and users cannot sort the information.
The number of columns and their sizes are determined by the column header. You can specify static column heads using CallbackArgs in the registration file or using the -commandHeading flag.

You can specify dynamic column heads using the -headingLine and -dataLine flags.

-**commandTitle**  <title>
   Title to use for the dialog box.

-**commandHeading**  <heading>
   If you are creating a table with more than one column heading, enclose the column headings in quotes and enter two spaces between the column headings. For example, "Status Color”. If you want to add space to a column heading, enter additional spaces. You will have to experiment with adding spaces to extend a column to the width you want. Remember to specify this information in the appropriate registration file. If you want to specify column headings on the command line, you can specify only one column heading. If no column headings are specified, appmon uses the size of the data to determine the column width.

-**dataLine**  <dataline>
   Which line of command output to begin using as output. A value of 3 for <dataline> will cause the first 2 lines not to be displayed.

-**headingLine**  <headingline>
   If you are creating a table with more than one column heading, you can create the column headings dynamically by specifying one of the lines of the command’s output with the -headingLine flag. For example, you could specify the first line of text as a column heading.

-**titleLine**  <titleline>
   Which line of command output to use as the title.

-**appendSelectList**
   Append nodename to shell_command.

-**appendSelectListToTitle**
   Append nodename to dialog box title.

-**timeout**  <seconds>
   The number of seconds to wait before sorting, printing, or saving results after a command is executed in batch mode. The default is two seconds.

-**sort**
   Sort results after command is executed.

-**print**
   Print results after command is executed.

-**save**
   Save results after command is executed.

-**exit**
   Exit appmon after command is executed, and after any -sort, -print, or -save option is executed. This feature is powerful in that it allows reports to be generated in batch mode. The configuration of a particular device can be sorted and saved to a file or printed. This can be done for many devices, without user intervention.

-**node**  <nodename>
   Use node <nodename> as target.
-maxLines  <lines>
    Change the number of maximum lines displayed. By default, the last 1000 lines are displayed.

Examples

If you want to encapsulate the output of the ovstatus command, use the following command in the appropriate registration file:

appmon -commandTitle "Status of NetView Products" -commandHeading Status -cmd /usr/OV/bin/ovstatus

Related Information

- See "OVwRegIntro" on page 591
- See "ovstatus" on page 360
event

Purpose
Sends an event to the trapd daemon.

Syntax
```
[-n num] [-s source] [-t hostname] [-u hostname] [-x]
```

Description
The event command sends an event to the trapd daemon. By default, this command sends an event with the following data values:

```
data = NULL
database = topo_db
descr = the event description
event = NUP_EV
node = none
num = 1
source = d
hostname = -u
```

Flags

- `-a data`
  Lists the character string containing any data that is associated with the event.

- `-b database`
  Specifies the path of the database that is associated with the event.

- `-d descr`
  Specifies the character string containing the description of the event. This value is in the `/usr/ov/log/trapd.log` file. Enclose the character string in quotes for multiple word descriptions. You can enter any text or description including format characters, such as `%s`, `%t`, and `
`.

- `-E event_number`
  Specifies the number of the event. See the `-e` option.

- `-e event`
  Specifies the character string containing the type of event. See the `-l` option.

- `-h node`
  Specifies the character string containing the node name associated with the event. This value appears in the `/usr/ov/log/trapd.log` file. The default for `node` is `none`, indicating the node on which the NetView program is running.

- `-l list`
  Lists all event types.

- `-n num`
  Sends an event the specified number of times.

- `-s source`
  Specifies one of the following characters containing the source of the event:
  
a Application
This value is in the `\usr\ov\log\trapd.log` file.

- **t** `hostname`
  Sends an event to `hostname` using TCP.

- **u** `hostname`
  Sends an event to `hostname` using UDP.

- **x** Hexdumps the packets sent to the trapd daemon.

### Implementation Specifics
The event command supports single-byte and multibyte character code sets.

### Related Information
See `netmon` on page 59.
evtstat

Purpose
Updates the status of nodes on a NetView map based on events received.

Syntax
 evtstat

Description
The evtstat command changes the status of nodes on the NetView map when an event, which has been appropriately configured by the trap application, is received by the management station. This application registers to receive events and changes the status of the corresponding object in the object database.

Implementation Specifics
This command can be run only from an application registration file (ARF).

Related Information
- See "trapd.conf" on page 703
findroute

Purpose
Determines IP routing using SNMP requests.

Syntax
findroute [-n] [-c community] [source] target

Description
The findroute command determines the route taken by IP packets between two nodes by using SNMP. An iterative process is applied in which the IP routing table of the source node is searched by SNMP requests to determine the next IP node in the route. This process repeats with the next node until the next node is equal to target. The default source is the local host.

For each node in the path, the findroute command prints the routing node name, the routing node IP address, the next node in the route, and the next node IP address. Names are determined from IP addresses as described in hosts.

The graphical user interface option equivalent to the findroute command is Test..Locate Route.

Note: The findroute command determines only one route between the source and target nodes. If a gateway supports multiple routes to the target node, the first route found is used. If a gateway in the route does not support SNMP, findroute cannot determine the complete route.

Flags
-`n` Does not truncate host names. By default, output host names are truncated to 20 characters.
-`c community`
  Overrides the community name, as configured in \usr\ov\conf\ovsnmp.conf, replacing it with `community`. See [ovsnmp.conf](#) on page 304.

Files
\usr\ov\conf\ovsnmp.conf
SNMP configuration

Related Information
- See [rnetstat](#) on page 673.
- See [netview](#) on page 65.
- See [ovsnmp.conf](#) on page 304.

Requests for commands
See the \usr\ov\doc directory for related RFCs.
**graph**

**Purpose**

Starts the NetView line graphing tool.

**Syntax**

```
graph [options] [node ...]
```

**Description**

The graph command starts a graphing utility window that can graph data collected by snmpCollect or real-time SNMP data. The graph command allows the user to graph real-time or historical data and to control many aspects of how the data is viewed.

The graph command supports integer, counter, gauge, and timeticks SNMP data types.

The references to menu in the following section pertain to the interactive graph modification after the graph is displayed.

**Flags**

The following options are available. The only required option is `-mib`.

- `-mib <line specifications>`
  
  This is the specification for each line to be graphed. It is a flexible but complex specification. It consists of an 8-term, colon-separated list of options in the following form, which is all one argument. The general form of the specification is:

  ```
  mibOID[mibLabel]:[instanceRE]:[instMatchOID][instMatchRE]:
  [instLabelOID][instLabelTrunc][mult]:[node[...]],...
  ```

  - **mibOID:** SNMP MIB Object in dotted notation (without instance). *mibOID* can be * if the -browse option is used, indicating to get all MIBs from database created by snmpCollect and perform no SNMP. See [snmpCollect](#) on page 680.

  - **miblabel:** Graph label to give the mibOID object (defaults to the textual last component of *mibOID* or line number when using extended data input.)

  - **instanceRE:** The regular expression of matching mibOID instances. This expression defaults to .*, which means to match any and all instances detected by the SNMP getnext command. The graph builds a menu of all possible instances by performing an SNMP getnext on the MIB variable to find all the instances for a given node. Uses extended regular expressions. Caution: the . character is a regular expression. If it really is the character desired, escape it using a backslash (\). The instance it is matched against begins with a number, not a dot. To match instances 15.6.80.3 and 15.6.80.4, the instanceRE would be 15.6.80.[34]. As a special case, if the instance does not begin with a dot, and contains only numbers and dots, it is assumed that an exact instance is used. In this case, no SNMP getnext command is sent.
instMatchOID:
The SNMP OID (minus instance) used with instMatchRE. The results of an
SNMP GET of this OID concatenated with an instance determined using
instanceRE is converted to a string and matched against instMatchRE. This
is used to limit the data that is actually graphed. See the following
information about instMatchRE.

instMatchRE:
A regular expression is matched against the results of an SNMP GET of
instMatchOID (default: match all). Both instMatchOID and instMatchRE
must not be defaulted to attempt matching. This feature is useful if there is
another object (usually in the same table as the object being graphed) that
determines whether a particular object should be graphed. As an example,
when graphing mibOID ifInOctets, only certain types of interfaces should be
graphed. The type of the interface is found in ifType. To access this MIB
variable, set instMatchOID to .1.3.6.1.2.1.2.2.1.3, the object ID of ifType. To
only graph interfaces of type ethernet-csmacd (ifType = 6), set instMatchRE
to 6. To graph all types but other (ifType = 0) and software loopback (ifType
= 24), set instMatchRE to the following:

\[([2-9]1|2[0-3]|2[5-9])3\]

If the value returned is an enumerated value, the enumeration is converted
to a number, not the enumerated string. This option uses extended regular
expressions.

instLabelOID:
The SNMP object ID (minus instance) queried to get the graph label for the
instance. The returned value is converted to a string and used in
conjunction with miblabel as the label for the corresponding instance of this
MIB variable. If the SNMP get command on this object ID fails or the object
ID is not supplied, the default instance label is the instance portion of the
full object ID. As an example, to use ifDescr as the instance label, set
instLabelOID to .1.3.6.1.2.1.2.2.1.2.

instLabelTrunc:
A regular expression string is used to truncate instance label. The instance
label is truncated at the first match of this string (if present). Leading and
trailing white space is always removed from instance labels, and labels of 0
(zero) value are not displayed. Uses extended regular expressions.

For example, the instance label can be:

"en0; Product: 802.3/ETHERNET
Manufacturer: 204491"

To remove the extraneous information, set instLabelTrunc to ";
This makes
"en0" the instance label.

mult:
A Multiplier of sampled value. Each sampled value is multiplied by this
value (if present) before being displayed, which enables data scaling. This
is the default value that the user sees in the View..Line Configuration menu.

node:
Space-separated list of nodes in either IP-address or node-name format. If
the list of nodes is not present, the node option defaults to the node list
passed at the end of the command line. If you are using data input
extensibility, this field can be used as a data name to cascade similar data
sets together. The node can be * if the -browse option is replaced. In this
case, the mibOID object will be retrieved from the snmpCollect database for all nodes. See "snmpCollect" on page 680.

Multiple colon-separated arguments are allowed, with commas separating each argument. The : and , characters are not valid in the colon-separated fields.

-browse
Show collected data. When in browse mode, no real-time SNMP data is obtained; only data in the snmpCollect database is displayed.

-poll <interval>
Use the specified interval to poll for data. This interval determines the polling frequency of SNMP data. If the interval value is 0 (zero), turn off polling. The interval is expressed as <number><unit>, where unit is s, m, h, d, w, or y for second, minute, hour, day, week, or year respectively. The interval is not used when data comes only from snmpCollect.

-title <title>
The title for the graph.

-units <units>
The units string is used to label the vertical axis of the graph. The defaults is no units.

Files

\usr\ov\conf\mibExpr.conf
\usr\ov\conf\snmpmib.bin
\usr\ov\conf\ovsnmp.conf

Related Information

- See the NetView help system and use the online graph help system.
- See "snmpodump" on page 687.
- See "snmpCollect" on page 680.
- See "mibExpr.conf" on page 51.
- Refer to \usr\ov\registration\c\ovip\nnm-ip.gph for examples of graph usage.
gtmd

Purpose

Receives generic topology information for the multiprotocol topology functions of the NetView program.

Description

The gtmd daemon provides a central repository of open topology information and is an SNMP interface. The gtmd daemon receives the traps generated by open topology discovery applications, proxy agents, and by open agents that implement the NetView topology MIB. The gtmd daemon stores an integrated view of all the topology objects in its management domain and communicates to xxmap to display the topology information. A complementary daemon, xxmd, must be configured whenever the gtmd daemon is configured.

As specified in the NetView topology MIB, the following types of traps can be sent by an agent implementing the topology MIB:

- New element
- Deleted element
- Operational state change
- Variable value change

Each of these traps triggers an operation in the gtmd daemon to update its current base of topology information. For each incoming trap, gtmd notifies the xxmap application.

By default, the gtmd daemon is not started when the NetView program is started. You can configure the gtmd (and xxmd) daemon through Server Setup indicating that it should be started by Server Setup. Users or an application can add the gtmd (and xxmd) daemon to the startup file without using NetView Setup, for example, by using the ovaddobj command. When you add gtmd (and xxmd) to the startup file, add the -Restart flag to the /usr/OV/registration/C/xxmap file. Add the -Restart flag to the following line in this file:

```
Command -Shared -Initial "${XXMapDir:-/usr/OV/bin}/xxmap"
```

The next example shows the line after you have changed it:

```
Command -Shared -Restart -Initial "${XXMapDir:-/usr/OV/bin}/xxmap"
```

If you reconfigure the gtmd and xxmd daemons, so that they do not start automatically, remove the -Restart flag from the /usr/OV/registration/C/xxmap file.

The otenable.bat file adds, in the \usr\ov\conf\ovsuf file, the gtmd and xxmd open topology daemons to be the list of daemons to start. After these daemons are added to the \usr\ov\conf\ovsuf file, you can take advantage of the open topology features and NVOT APIs. The otdisable.bat file removes, from the \usr\ov\conf\ovsuf file, the gtmd and xxmd open topology daemons from the list of daemons to start.

Parameters

- `-l` Specifies the log file name
- `-b (integer)` Specifies the buffer link, which is the number of buffers that gtmd uses to save information before sending it to xxmap
-f (integer)
    Specifies the database flush time, which is the frequency in seconds that
gtmd should flush its memory to the database on disk

-t
    Specifies the trace file name

Related Information
See "xxmap" on page 711
ipmap

Purpose
Displays a map of the network discovered by the netmon daemon, using the
NetView graphical user interface.

Syntax

```
ipmap [-d] [-s on|off|connStatusOn|connStatusOff|compoundNodeOn]
        [-b batch_size] [-D level] [-l label_prefix] [-L label_suffix]
```

Description
The ipmap command reads IP topology data from the ovtopmd database and
listens for IP topology events, which it uses to display the IP network topology.

The IP topology is displayed in a hierarchy of submaps that is initially four levels
deep: internet level, network level, segment level, and node level.

**Map Synchronization:** When ipmap is started it begins its synchronization phase.
While ipmap is synchronizing, a synchronizing message is displayed by the
graphical user interface on the status line of the displayed submaps. During
synchronization, ipmap updates the IP map with the ovtopmd database topology
data by creating symbols for new network objects. During this time, ipmap cannot
respond to some NetView requests, and you cannot access certain functions.

For example, the NetView program restricts you from deleting symbols and objects
while any application is synchronizing. Also, ipmap will not be displayed in the list of
applications on dialog boxes until it has completed its initial synchronization. The
manage object and unmanage object operations are accessible, but ipmap will not
act on those operations until it has finished its initial synchronization. The
NetView program will manage the objects as you request, but the
ovwManageObjects event that was sent to ipmap is queued until it has finished
synchronizing. When ipmap has synchronized, it manages incoming NetView events
and SNMP traps from the ovtopmd daemon.

The ipmap command will also display a synchronizing message during operations
that can take a long time. This includes configuration changes, delete operations,
manage and unmanage operations. The same restrictions apply to these
intermittent synchronization phases. However, ipmap will be displayed in the list of
applications in the add dialog boxes. If you attempt to add an object while ipmap is
synchronizing, ipmap will not respond to NetView events.

You might see the synchronizing message flash occasionally while netmon is
discovering the network. This is expected behavior. The short periods of
synchronization are a result of symbols being deleted and re-added as the netmon
daemon moves them from one segment to another.

**Map Editing:** The ipmap command supports map editing by the user. This is
accomplished through the graphical user interface. You can modify the map through
the graphical user interface in several ways, including: adding symbols, deleting
symbols, and cutting and pasting symbols. As mentioned above, these operations
are not accessible during map synchronization.
The addition of symbols includes both icon symbols and connection symbols, which are accessed through different menu items in the graphical user interface. Each submap type has a different set of symbol types that you can add. This is explained in detail in the following submap sections, "Connections" on page 40 through "Interface Symbol Types" on page 43.

The ipmap command supports the deletion of all symbols from the map except the following:
- The internet symbol that is placed in the root submap
- Certain interfaces discovered by netmon

Deleting symbols from a map does not guarantee that the corresponding objects will be deleted from the topology database. Objects are deleted from the topology database when all symbols for that object have been removed from maps. Therefore, if you choose to delete an object so that it is rediscovered by the netmon daemon, such as a network, you must delete that object from all maps.

The cut and paste operation is supported by the ipmap command in the internet submaps. Although you can access and perform the cut and paste operations in all the submaps, the ipmap command responds to the cut and paste operations only on the internet submaps. In other submaps, the symbol remains in the user plane after the paste operation. The details of these operations are explained in the following submap sections, "Connections" on page 40 through "Interface Symbol Types" on page 43.

### Flags

- **-b** Specifies the number of symbols to be batched for creation. By default, this value is 120 but it can be changed to either improve performance or improve user feedback. By setting this value to a smaller number, you will see smaller, more frequent updates to the map when symbols are added. This benefit causes a decrease in performance, however, because it requires more context switches. By setting this value to a larger number, you will see larger, less frequent updates. As a consequence, you can draw large maps more quickly.

- **-d** Disables ipmap by default. When the ipmap command is disabled, it is executing but is dormant. It will not update the map with any topology data. You can create a map in which ipmap is enabled, but the default behavior is for it to be disabled.

- **-s** Configures status defaults. The valid values are on, off, connStatusOn, connStatusOff, and compoundNodeOn.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>Set the status of interfaces</td>
</tr>
<tr>
<td>off</td>
<td>Do not set any IP status</td>
</tr>
<tr>
<td>connStatusOn</td>
<td>Connection status set the same as underlying interface</td>
</tr>
<tr>
<td>connStatusOff</td>
<td>Connection status is not set</td>
</tr>
<tr>
<td>compoundNodeOn</td>
<td>Status of node symbols reflect the status of all components</td>
</tr>
</tbody>
</table>

- **-D** Demands creation level for submaps. By default, the ipmap command is configured to place all submaps and symbols in the map as soon as they are
discovered. This guarantees the existence of those symbols for user operations and other applications. For large networks, however, this can be quite costly. If you want to reduce the amount of system resources that the NetView program and ipmap use, you can configure ipmap to create submaps only on demand. The initial level of submaps created is set by this option. The valid values are 3, 2, 1, and 0 (zero).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Create only the internet submap initially</td>
</tr>
<tr>
<td>2</td>
<td>Create the internet submap and all network submaps</td>
</tr>
<tr>
<td>1</td>
<td>Create the internet, network and all segment submaps</td>
</tr>
<tr>
<td>0</td>
<td>Create all submaps</td>
</tr>
</tbody>
</table>

Symbols of objects that do not yet have a submap created are indicated by a special suffix attached to their label. By default the suffix is an asterisk (*), but this can be changed by using the -l and -L options, which are explained in [flags -l and -L](#).

When the submap demand level is changed from the default there are certain caveats that must be called out. First, because some of the symbols have not been placed on the map yet, the user cannot locate all objects that have been discovered. Only those objects that have symbols on the map can be found using the Locate function.

If the submap demand level is set to 3 (create only the internet submap), you cannot successfully partition the internet submap using cut and paste operations as described in [Internet Submap](#) on page 40. If this is attempted, ipmap will not maintain the connections between existing symbols and the new container symbols unless the networks being moved have already been opened; therefore, they have created a submap.

If the submap demand level is set to 1 (create submaps down to the segment level) then the user might be unable to cut and paste nodes between segments of a network.

- **-l** Label prefix for symbols of objects with no child submap. The default is the empty string. This option is used in conjunction with the -D option.

  **Note:** Although interface objects do not have child submaps, their labels are not altered using this prefix, because ipmap will not create a child submap for an interface.

- **-L** Label suffix for symbols of objects with no child submap. The default is the string (*). This option is used in conjunction with the -D option.

  **Note:** Although interface objects do not have child submaps, their labels are not altered using this suffix, because the ipmap command will not create a child submap for an interface.

- **-o <number>**
  Specifies the number of seconds to wait after the first object is cached. The default is 3 seconds.

**The ipmap Configuration:** You can configure the ipmap command by using the Edit Properties map tab panel in the graphical user interface menu. The following items are configurable in the dialog box:
• Automatically Manage Objects
  Indicates whether ipmap should create symbols on this map. Select TRUE for
  ipmap symbols; select FALSE for no symbols. This option can only be accessed
  when a map is being created. After the map is created, this option is grayed.
  When this option is set to FALSE for a map, ipmap will run, but it will not update
  the map. If a new map is opened where ipmap is enabled, it will begin updating
  that map.

• Update Status for Objects
  Indicates whether ipmap should set the status of objects on the map. Select
  TRUE if ipmap should set the status. Select FALSE if you do not want any IP
  status on this map. You can change this selection at any time.

• Propagate Status of Interface Cards to Connections
  Indicates whether connection lines should show interface status. Select TRUE for
  the status of connection lines that are set to be the same as the interface that
  the connection represents. Select FALSE for connections to show only a normal
  status. You can change these options at any time.

• Exclusively Set the Status for Nodes
  Indicates whether ipmap should explicitly set the status of node symbols on the
  map. Select TRUE, for ipmap to set the status of node symbols based on the IP
  context in which the node symbol resides. For example, if the gateway has two
  interfaces (one interface on network A and the other on network B) and the
  interface on network A is down, the node symbols for the gateway that are in the
  context of network A (the network and segment submaps for network A) are set
to critical status. The symbols of the gateway in the context of network B remain
in normal status. The gateway symbol on the internet submap becomes marginal,
because it is not in either context. The status of open component symbols in the
node submap are not considered for status propagation.
  Select FALSE to set the status of all components of a node to be propagated to
the node symbol according to the propagation rules set for the map. In the case
of the previous example, the status of all symbols for this gateway would be set
to marginal.

**Connections**: The connection symbols that are created by ipmap are symbols of
an interface object. For example, in the internet submap, a gateway is connected to
a network. The connection between them is a symbol of the gateway’s interface
that is attached to the network. This facilitates the display of interface status on
higher levels of the map. When a gateway’s interface is not functioning, you can
determine to which network the bad interface is connected. This is configurable.
See the information about [ipmap Configuration](#) on page 39.

For example, a connection that is not a symbol of an interface object, is a serial link
between gateways. See [Serial Networks](#) on page 42 for more details.

**Internet Submap**: The internet submap is accessed by exploding the internet
symbol that appears in the root submap. This submap represents the highest level
view of the network as it contains gateways, networks, and the connections
between them.

Only gateways and networks can be added to this submap. Other symbol types are
not recognized by ipmap and are ignored. Gateways have the isRouter capability
field set to TRUE, and the networks must have both the isIP and isNetwork
capabilities set to TRUE.
The internet submap can be partitioned into a hierarchy of internet submaps, thus adding depth and breadth to the default ipmap submap hierarchy. This is particularly useful when the default internet submap has many symbols or when you want the map to represent a logical or physical hierarchy (geographical regions or administrative domains). It is accomplished by creating new internet submaps and using the cut and paste operations to move symbols from the original internet submap to the new ones.

To create internet submaps, the first step is to add either an internet symbol in the network class or a symbol in the location class to the internet submap. Wait for ipmap to complete its synchronization phase before adding these symbols. When the new symbol is added, ipmap will create a new internet submap under that symbol. An arbitrary number of levels can be created here by adding new internets to the newly created internet submaps.

After all of the new internets and corresponding submaps have been added, the next step is to move symbols into the new submaps. This is done by cutting symbols from the original internet submap and pasting them into one of the new submaps. As the symbols are moved in this manner, ipmap will maintain connections automatically. If there are multiple connections between two symbols, the graphical user interface will automatically create a compound connection submap showing all of the connections between the symbols.

When new gateways or networks are discovered by the netmon daemon, ipmap will add them to the original internet submap. Then you can move them to the appropriate internet submap. However, if you want to add a gateway or network, add these to any of the internet submaps.

These map modifications are specific to this map. If the user creates a new map after adding new internet symbols and submaps, the ipmap command will create the new map with only the default internet submap. Any modifications of this type are not considered topology data and thus are not stored in the ovtopmd database.

**Network Submap**: The network submap is a representation of an IP network, which contains a collection of segments, connector devices (gateways, bridges, and hubs) and the connections between them.

Connectors have the isConnector capability set to TRUE. This includes all symbols in the connector class. Segments have the isSegment capability set to TRUE. Segment symbols are in the network class and include bus, star, and ring segments.

**Segment Submap**: The segment submap displays an individual segment of the network. There are several types of segments supported: bus, star, and ring. Each type uses the corresponding layout algorithm provided by the graphical user interface. The symbols allowed in this submap are those with the isNode capability set to TRUE. This includes all symbols in the computer and connector classes. If symbols from other classes are added, they are ignored by ipmap.

The user can move nodes between segments of a network. This is useful if it is necessary for the user to manually partition a network into segments and is required if some of the bridges on the network do not support SNMP. To move a node from one segment to another, cut the node symbol from one segment submap then paste it to the new submap. The ipmap command will check to ensure that the node being pasted is actually from the same network. If it is, then ipmap will accept
the new node symbol and move it to the application plane. If it came from a
different network, ipmap ignores the new symbol.

**Node Submap:** The node submap is the lowest level submap provided by ipmap. When created it contains only interface icon symbols for the interfaces discovered by the netmon daemon. Other applications, however, can add additional components to a node.

The only symbol that ipmap recognizes in this submap is the IP interface in the card class.

**Serial Networks:** Serial networks are treated as a special case by the ipmap command. A serial network will be displayed in one of two ways:

- When a gateway is first discovered with a serial interface, the serial network connected to that interface is displayed as a network symbol like any other network.
- When the user manages that network and the node on the other side is discovered, the network symbol is replaced with a connection symbol between the two gateways. The connection symbol is a symbol of the network object so that connection can be exploded into the network submap.

At the network submap level, serial segments are treated in the same manner as serial networks on the internet submap. If a serial segment is detected between two connectors, it is replaced by a connection between those connectors. That connection is a symbol of the segment object, so it explodes into a segment submap.

**Default Symbol Types:** When ipmap adds a symbol to a submap it sets an initial default symbol type to be associated with the object in that submap. The user may change that symbol later without affecting the behavior of the IP topology or ipmap. See “OVwRegIntro” on page 591 for more details about symbol type and field registration.

**Internet Symbol Type**
The default symbol type for the Internet Submap is Network:Internet.

**Network Symbol Type**
The default symbol type for networks is Network:IP.

**Segment Symbol Type**
The default symbol type for segments will be one of the following based on the capabilities associated with the segment:

- Network:Bus
- Network:Star
- Network:Token
- Network:FDDI Ring

See “ovtopmd” on page 363 for more information on the capabilities associated with a segment.

**Node Symbol Types**
The default symbol type for nodes will normally be determined based on a mapping from the SNMP system.sysObjectld associated with the node, as determined by the oid_to_sym configuration file. For nodes without a sysObjectld, the default symbol type is determined based on the field properties associated with an object. The symbol type will be one of the following:
Nodes that have no particular IP topology behaviors and no SNMP sysObjectID will have a default symbol type of **Computer:Generic**. See “ovtopmd” on page 363 for more information on the properties associated with nodes.

**Interface Symbol Types**

The symbol type for interfaces is based on the SNMP ifType associated with the interface and will be one of the following:

- Cards:Generic
- Cards:IP
- Cards:Serial
- Cards:Star Lan
- Cards:FDDI
- Cards:Token Ring
- Cards:X.25

If no SNMP ifType is associated with the interface but the interface has an IP address, the symbol type will be Cards:IP; otherwise the symbol type will be Cards:Generic.

**Application Integration Information:**

**Database Fields**

For each type of object represented in the submaps created by ipmap, there are a number of fields maintained in the OVW object database. These fields are described in ovtopmd.

**Submap Type Values**

The OVwSubmapInfo structure contains a field named submap_type. See the ov\ovw.h header file for more information. For submaps created by ipmap, this field is set to one of the following values:

<table>
<thead>
<tr>
<th>Submap Type</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet submap</td>
<td>2</td>
</tr>
<tr>
<td>Network submap</td>
<td>3</td>
</tr>
<tr>
<td>Segment submap</td>
<td>4</td>
</tr>
<tr>
<td>Node submap</td>
<td>5</td>
</tr>
</tbody>
</table>

**Files**

`\usr\ov\conf\C\oid_to_sym`

SNMP sysObjectID to symbol type mapping

**Related Information**

- See “netview” on page 63
- See “ovwdb” on page 464
- See “ovtopmd” on page 363
- See “oid_to_sym” on page 273
- See “netmon” on page 59
- See “OVwRegIntro” on page 591
loadhosts

Purpose

Loads hosts to the IP topology database.

Syntax


Description

The loadhosts command loads (adds) hosts to the IP topology database maintained by the ovtopmd daemon and used by the netmon daemon and the ipmap command. Lines are read from standard input and are in the format used by winnt35\system32\drivers\etc\hosts.

For each input line, loadhosts uses the IP address and host name to add an entry to the IP topology database. If no entry exists that matches the host name, an entry is created. If an entry already exists for that host name, the IP address is added as an additional interface for the input host name. Entries with the same host name and multiple IP addresses are assumed to be gateways.

The network mask is automatically generated based on the class of the IP address, and the status of the new entry is set to Up.

Loadhosts can also add new interfaces as Unmanaged by using the ! token at the beginning of an entry. This option overrides the -d switch (which marks newly-added interfaces as Down). When using the ! token, the netmon -l switch should be set to create new networks as managed.

Flags

-C contact
Sets the sysContact field for all added nodes to contact.

-d Marks newly added interfaces as Down.

-D descrip
Sets the sysDescrip field for all added nodes to descrip.

-L location
Sets the sysLocation field for all added nodes to location.

-m mask
Uses mask as the IP subnet mask instead of determining it based on the class of the IP address of new nodes.

-M Creates physical addresses. As each IP address is processed, an algorithm creates an IP address for the interface. This option is normally used only for testing purposes.

-P Masks new nodes as supporting the SNMP protocol. This option is normally used only for testing purposes.

-S stime
Delays stime seconds between the addition of new nodes or interfaces. The default is no delay.
-t Traces input lines as they are read. This option is useful for monitoring the progress of the operation.

-v Activates verbose mode: lists messages describing the actions taken on the database.

Examples

You can load the gateway gw1 with IP addresses 15.2.112.1 and 15.2.40.1, and the node node1 with IP address 15.2.112.96 by entering:

```bash
loadhosts -m 255.255.240.0
15.2.112.1 gw1
15.2.40.1 gw1
15.2.112.96 node1
```

ICMP Echo requests verify the IP addresses, and the subnet mask is set to 255.255.248.0.

You can also load host information using `loadhosts <host.txt`. The host.txt file contains host information.

You can add new interfaces as unmanaged by entering:

```bash
!14.8.2.100 myrouter.ibm.com myroute
```

Files

```
\usr\ov\databases\openview\topo*
  The ovtopmd database directories
```

Related Information

- See "ipmap" on page 37.
- See "netmon" on page 58.
- See "ovtopmd" on page 363.
- See "ovtopodump" on page 366.
- See "netview" on page 65.
loadmib

Purpose

Loads and unloads SNMP MIBs

Syntax

loadmib [options]

Description

The loadmib command loads an SNMP MIB (Management Information Base) in the Internet SMI (Structure of Management Information) format for use by the NetView graphical user interface in making SNMP requests. The loadmib command maintains the ASCII file `\usr\ov\conf\snmpmib`, which is a collection of MIB modules, and compiles it to produce `\usr\ov\conf\snmpmib.bin` for use by NetView SNMP applications.

The loadmib command has both a command-line mode, which is well suited for installation scripts, and an interactive mode, which can either run standalone as a windows application or can be invoked from the NetView graphical user interface with the Tools..MIB..Tool Builder. See `netview` on page 65.

Flags

The loadmib command supports the following options:

- **-noevent**
  Do not send a MIB changed event (SNMP Trap) to alert NetView applications to reread the MIB database.

- **-mibDB mib-dbname**
  Use `mib-dbname` instead of default MIB Database `\usr\ov\conf\snmpmib`

- **-mibDir directory**
  Use `directory` instead of the default directory `\usr\ov\snmp_mibs` as the MIB file selection directory. (interactive mode only)

- **-compileOnly**
  Only regenerate binary form of MIB Database. (non-interactive mode only)

- **-load mib-file**
  Load `mib-file` into MIB Database. (non-interactive mode only)

- **-unload mib-module**
  Unload `mib-module` from MIB Database. (non-interactive mode only)

- **-replace**
  Force reload of MIB if already loaded. (non-interactive mode only)

- **-silent**
  Do not print error results to stderr. (non-interactive mode only)

Diagnostics: The following exit codes are returned for the non-interactive mode:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The operation was successful.</td>
</tr>
<tr>
<td>1</td>
<td>The command line usage was not valid.</td>
</tr>
<tr>
<td>2</td>
<td>A syntax error was detected in the MIB file.</td>
</tr>
</tbody>
</table>
The MIB file was already loaded and `-replace` was not used.

MIB module specified by `-unload` was not loaded.

MIB file cannot be accessed.

MIB file specified with `-load` is not a file.

Error accessing MIB Database.

Error creating MIB Database.

Error creating compiled form MIB Database.

Error creating working temporary MIB Database.

loadmib interrupted by signal.

loadmib out of memory.

**Examples**

You can start loadmib interactively by entering the following command:

```
loadmib
```

You could also select Load/Unload MIBs from the Options menu in NetView graphical user interface. The graphical user interface option equivalent to the loadmib command is Tools..MIB..Load.

You can load the MIB file `\user\data\corp.mib` using the non-interactive mode by entering the following command:

```
loadmib -load \user\data\corp.mib
```

**Files**

- `\usr\lov\conf\snmpmib`
  - Text version of MIB Database
- `\usr\lov\conf\snmpmib.bin`
  - Compiled version of MIB Database
- `\usr\lov\snmp_mibs`*
  - Standard and Vendor MIBs
- `\usr\lov\tmp\mib`*
  - Temporary work files

**Related Information**

See "netview" on page 65.
**Purpose**

Provides the format for the NetView local registration file (LRF).

**Syntax**

The lrf lines consist of fields separated by colons. Each line has a predefined constant number of fields that it must contain. Fields for which defaults are defined can be omitted, but all separating colons are required. Empty lines and lines starting with a # symbol are treated as comment lines and are ignored. All characters appearing after a # on a noncomment line are also treated as comments and are ignored.

All fields, unless otherwise stated, cannot contain certain characters. Normally, a field cannot contain control characters (ASCII values less than 32), blank characters, colons (:), commas (,), backslashes (\), number signs (#), or exception characters (characters with ASCII values greater than 126). For any field that does enable exception characters, the exception character cannot be used literally, but must be encoded. The encoded form is \nnn where nnn is a 3-digit octave representation of the ASCII value of the exception character.

**Description**

The lrf file is read by the commands ovaddobj and ovdelobj to register or deregister an agent. The lrf file conveys the information to the ovspmd daemon to start the daemon program as a daemon.

**First Line:** The first line of an lrf file is required and must contain the following fields in the format

```
Name:Path:,
```

where:

**Name**

Specifies the official name by which this agent is known to ovstart, ovstop, and ovstatus commands and the ovspmd daemon. This name must be locally unique and, if the name is changed, the agent must be deleted with ovdelobj before changing the name and must be added under its new name with ovaddobj. This is a required field. There is no default.

**Path**

Specifies the absolute path name of the executable agent program. This is a required field. There is no default.

**Second Line** The second line of an lrf is required and must contain the following fields in this format:

```
Initial_Start_Flag:Dependencies:Arguments:Behavior:Timeout:
```

Where:

**Initial_Start_Flag**

Must be either OVs_YES_START or OVs_NO_START. If it is OVs_YES_START, the agent will be started automatically when ovstart is run with no arguments. If it is OVs_NO_START, it will be started only if ovstart is explicitly run with the
name (see “Name” field) of the agent as an argument, or if another agent
depends on this one. If not specified, Initial_Start_Flag will default to OVs_NO_ 
START.

**Dependencies**

Is a comma-separated list of other agents (specified by their "Name" fields) that
must already be running before this agent can be started. The ovspmd
command will start agents in the correct order to satisfy these dependencies. See [ovspmd](#) on page 354 and [OVsPMD_API](#) on page 356.

By default, an agent has no dependencies; it can be started even if no other
NetView components are running.

**Arguments**

Is a comma-separated list of command line arguments with which the ovspmd
daemon will start the agent. By default, the agent program is started with no
arguments.

**Behavior**

Describes whether the agent has been integrated with the ovspmd daemon
using the OVsPMD_API. Behavior must be OVs_WELL_BEHAVED,
OVs_NON_WELL_BEHAVED, or OVs_DAEMON as described in the following
list:

**OVs_WELL_BEHAVED**

The agent is fully integrated with the ovspmd daemon. The ovspmd
daemon does not expects the agent to go into the background of its
own accord, but to let ovspmd start it in the background as its child.
The ovspmd daemon also expects the agent to establish
communication with it using the functions defined in OVsPMD_API,
running only after it has sent a SUCCESS response and exiting when
ovspmd sends an EXIT command.

**OVs_NON_WELL_BEHAVED**

The agent is not integrated with the ovspmd daemon. The ovspmd
daemon does not expect the agent to go into the background of its own
accord, but lets ovspmd start it in the background as its child. The
ovspmd daemon assumes that the agent is running after the value of
**Time-out** seconds has elapsed and the ovspmd daemon terminates it.

OVs_NON_WELL_BEHAVED managers must set up a signal() routine
to handle termination through SIGABRT if the managers do not exit
within the configured time-out.

**OVs_DAEMON**

The agent is not integrated with the ovspmd daemon and runs as a
daemon in the background. Because the ovspmd daemon cannot
manage a process that is not its child, it can start such an agent but
cannot terminate it or report meaningful status about it.

If the definition does not match the actual behavior of the agent, the
ovspmd daemon treats the agent as the agent is configured but does
not adapt to its actual behavior. If not specified, the Behavior field
defaults to OVs_NON_WELL_BEHAVED.

**Time-out**

Must contain an integer that is interpreted as a number of seconds. If an agent
is configured as OVs_NON_WELL_BEHAVED, the ovspmd daemon will
assume the agent program has fully initialized and is running after **Time-out**
seconds have elapsed. The ovspmd daemon will then start other agents that
depend on it. When ovspmd stops the agent with SIGTERM and the agent does not exit within Time-out seconds, the ovspmd daemon will stop the agent with SIGKILL.

If an agent is configured as OVs_WELL_BEHAVED and sends a DONE response but does not actually exit within Time-out seconds, ovspmd will stop the agent with SIGTERM (and will later terminate the agent with SIGKILL if it still does not exit). The default Time-out value is five seconds.

Examples

First Line:
netmon:usr\ov\bin\netmon.exe:

The official name of the agent is netmon, and the executable for netmon is \usr\ov\bin\netmon.exe.

Second Line:
OVs_YES_START:ovtopmd,trapd,ovwdb:-P:OVs_WELL_BEHAVED:15:

The netmon agent is started by the ovspmd daemon by default. It depends on the agents ovtopmd, trapd, and ovwdb being started before it. The netmon agent should be started with the -P option. It is a Well-Behaved agent. Its timeout is 15 seconds.

Related Information

- See "ovaddobj" on page 277
- See "ovdelobj" on page 279
- See "ovspmd" on page 354
- See "ovstatus" on page 360
- See "ovstart" on page 358
- See "ovstop" on page 361
- See "OVsPMD_API" on page 356
mibExpr.conf

Purpose

Provides NetView MIB expression configuration.

Description

The \usr\ov\conf\mibExpr.conf file associates a name with an SNMP MIB expression. This expression is used in applications such as the data collector and snmpCollect.

The expressions are entered in postfix notation.

The form of this file is: name description expression.

The description may be separated into lines using the carriage-return character (15, ctrl-M) as the line separator. Fifty characters per line is the normal maximum. An expression can be any combination of values or operators in postfix notation. The infix expression \( \frac{A}{B+C} \) would be \( ABC+/\) when using postfix. Operators can be any of the following symbols:

\[ \text{Operators: } +, -, *, / \]

The value field must be a MIB variable or number. MIB variables always start with a period (\( . \)). If a MIB variable ends in a period, it means that an instance is to be appended by the application using the expression.

Examples

```plaintext
# sample \usr\ov\conf\mibExpr.conf file
#
# Calculate the percentage deferred on any interface
#
# If%deferred \
"packets deferred/packets transmitted" \
.1.3.6.1.4.1.11.2.4.1.1.1.1.
.1.3.6.1.4.1.11.2.4.1.1.1.2. \
/ 100 * 
#
# Convert numUsers to a negative number, 
# so that the data collector 
# can generate a threshold on when the number of users drops 
# below a certain value. 
# 
# negNumUsers \

*Negate value of numUsers MIB variable so the data 
collector \( M \) 
can detect when this value drops below a certain value. \( M \) 
Note that the threshold must be set negative, so if you 
want \( M \) 
to know when there are less than 5 users, set the data 
collector\( M \) 
threshold to generate an event when this expression exceeds 
\( -5. \) 
.1.3.6.1.4.1.11.2.3.1.1.2.0 -1 *
```

Files

\usr\ov\conf\mibExpr.conf
Warning
Syntax errors, in this file, can cause data collection or graphing to not function. Deleting entries that the data collector or grapher require will also cause errors in those tools.

Related Information
See "snmpCollect" on page 680.
mibform

Purpose
Gets and displays MIB objects in simple form format.

Syntax

mibform -obj objlist -node nodeName [-c community] [-d] [-appname applicationName] [-t title]

Flags

-node
  Specifies the name or IP address of the device to query

-obj
  Specifies a list of object IDs, separated by commas, where:
  objlist :: objSpec[,objSpec]...
  objSpec :: [label=]objId[:width]
  objId :: object ID of the MIB variable
  label :: label for the MIB variable
  width :: display width of the MIB variable

-c
  Specifies the SNMP community name

-d
  Dumps SNMP packets

-appname
  Specifies a unique name of the MIB application

-title
  Specifies the title for the MIB application window

-inst
  Specifies one of the following:
  • A single instance ID to be applied to all object IDs specified by -obj. The default is 0.
  • A comma separated list of instance IDs to be applied respectively to the object IDs specified by -obj.

  The number of instances specified must equal one or the number of object IDs specified by the -obj parameter.

Examples

mibform -obj sysDesc=.1.3.6.1.2.1.1.1.1.,
  sysObjectID=.1.3.6.1.2.1.1.2.,
  sysUpTime=.1.3.6.1.2.1.1.3,
  sysContact=.1.3.6.1.2.1.1.4,
  sysName=.1.3.6.1.2.1.1.5,
  sysLocation=.1.3.6.1.2.1.1.6,
  -node aixnmmjad

  The output of this command is similar to the following:
User Specified MIB Description for aixnmjad

sysDesc : IBM RS/6000
: Machine Type: 0x0101 Processor id: 000009317500
: The Base Operating System AIX version: 03.02.0000.0000
: TCP/IP Applications version: 03.02.0000.0000
sysObjectID : .iso.org.dod.internet.private.enterprises.ibm.3.1.2.1.1.2
sysUpTime : (234864083) 27 days, 4:24:00.83
sysContact : Judith Dietzmaster
sysName : aixnmjad
sysLocation : UI25
mibserver

Purpose
The mibserver daemon is the Tivoli NetView MIB server.

Syntax
mibserver

Description
The mibserver daemon runs the MIB server. The mibserver daemon must be
started on the NetView server in order to run the MIB browser in the NetView Web
console.

Dependencies
By default, the mibserver daemon is not registered with ovspmd. It can be
registered in either of two ways: using the ovaddobj command-line option, or
through the Server Setup Application.

The mibserver daemon is dependent on other daemons (webserver, ovtopmd,
trapd, ovwdb) and, when registered with ovspmd, is started and stopped
automatically with the other NetView daemons through the Server Setup
application, or by using the ovstart and ovstop commands.

The ovspmd daemon uses the spmsur.exe application to launch the mibserver
daemon.

Dependencies
The mibserver daemon is under the control of ovspmd. It is dependent on other
daemons (webserver, ovtopmd, trapd, ovwdb) and is started and stopped
automatically with the other NetView daemons through the Server Setup
application, or by using ovstart and ovstop commands.

By default, the mibserver daemon is not registered with ovspmd. It can be
registered in either of two ways: using the ovaddob command-line option, or
through the Server Setup Application.

The mibserver daemon uses the spmsur.exe application.

Related Information
• See “ovaddobj” on page 277
• See “ovdelobj” on page 279
• See “ovspmd” on page 354
• See “ovstatus” on page 360
• See “ovstart” on page 358
• See “ovstop” on page 361
• See “polld” on page 671
• See “spmsur.exe” on page 694
mibtable

Purpose

Gets and displays an arbitrary MIB table.

Syntax

\texttt{mibtable -table objID -fields fieldList -node nodeName [-c community] [-d] [-appname applicationName] [-t title]}

Flags

- \texttt{-table}
  - Specifies the object ID of the table entry

- \texttt{-fields}
  - Specifies a list of field IDs, separated by commas, where:
    \texttt{fieldList :: fieldSpec[,fieldSpec]...}
    \texttt{fieldSpec :: [label=]fieldId[:width]}
    \texttt{fieldId :: object sub-ID of the field}
    \texttt{label :: column label for the field}
    \texttt{width :: column width for the field}

- \texttt{-node}
  - Specifies the name or IP address of the device to be queried

- \texttt{-c}
  - Specifies the SNMP community name

- \texttt{-d}
  - Dumps SNMP packets

- \texttt{-appname}
  - Specifies a unique name of the MIB application

- \texttt{-title}
  - Specifies the title for the MIB application window

- \texttt{-inst}
  - Specifies a list of instance identifiers, separated by commas, that specify the rows to be retrieved from the table. The default is to return all rows. Instance identifiers can be specified as numeric values, for example, 1,3,5, or character strings, for example, ps,/usr, where the ASCII equivalent for the characters is used as the instance ID. Also, an asterisk (*) can be used as a pattern-matching character at the end of an instance.

Examples

The following command:

\begin{verbatim}
  mibtable -table .1.3.6.1.2.1.2.2.1
  -fields ifType=3:22,ifMtu=4:11,ifSpeed=5:10,ifPhysAddress=6:18
  -node aixnmjad
\end{verbatim}

Produces output similar to the following:

\begin{verbatim}
  User Specified MIB Table for aixnmjad
  ifType ifMtu ifSpeed ifPhysAddress
  softwareLoopback 1536 0
  iso88025-tokenRing 1492 16000000 10 00 5A B1 6C 39
\end{verbatim}
mib2trap(1)

Purpose
Automatically customizes trap definitions using a MIB definition file.

Syntax
mib2trap mib_file output_file

Description
The `mib2trap` command retrieves trap definitions special comments in the specified MIB definition file and creates a bat file that contains a series of `addtrap` commands. You can execute the bat file to add the new trap definition to the `conf\trapd.conf` file.

Note: If you receive error messages relating to an unknown enterprise, load the MIB file first.

The MIB definition file can have comments in the following form:

---#keyword

The `mib2trap` command recognizes the following keywords:

---#SUMMARY "...
---#ARGUMENTS {#, #,..., #}
---#SEVERITY INFORMATIONAL|WARNING|MINOR|MAJOR|CRITICAL
---#GENERIC -1|0|1|2|3|4|5|6
---#CATEGORY "IGNORE"|"LOGONLY"|"Error Events"|
---#SOURCE_ID "T"|"N"|"A"|... (= Trapd, Netmon, Agent, ...)
---#STATE DEGRADED/OPERATIONAL/NONOPERATIONAL

Following is an example of an entry in a MIB definition file:

```
serverDown TRAP-TYPE
ENTERPRISE netware-GA-alert-mib
VARIABLES {
    osName, osLoc, tiTrapTime, tiEventValue,
    tiEventSeverity, tiUserName, tiStationNum
}
DESCRIPTION
"This Netware server is going down."
---#SUMMARY "%s is going down"
---#ARGUMENTS {0}
---#SEVERITY MINOR
---#STATE NONOPERATIONAL
::= 2
```

Flags

- **mib_file**: Specifies the input MIB definition file
- **output_file**: Specifies the output .bat file, which contains the addtrap commands
Related Information

- See "addtrap" on page 20.
- See "trapd.conf" on page 703.
netmon

Purpose

Discovers and monitors nodes on the network.

Syntax

```
```

Description

The netmon daemon attempts to discover nodes on the network. After it discovers a node, it polls the node regularly to check for status, topology, configuration, and threshold changes.

The netmon daemon discovers and polls only the nodes in the management region. The initial management region is the network or networks containing the node on which the netmon daemon is executing.

Netmon will discover unnumbered interfaces and create the serial links. This option is enabled using the -u switch and is set by default. When using the default local discovery mode, netmon will discover the router at the other end of the unnumbered serial link as unmanaged. You must manually manage it in order for the link to be created. If you experience difficulty creating the link, check the next hop configured on each router. For Cisco routers this can be seen in the output of a demand poll. Check that the IP name of the next hop can be resolved to a correct IP address of the router using nslookup. For more details see the Diagnosing and Solving NetView Problems chapter in the Tivoli NetView for Windows User’s Guide.

The netmon daemon supports the Hot Standby Router Protocol (HSRP). HSRP is where, an HSRP IP address of an active router becomes unavailable and a second router takes over this HSRP IP address. Netmon must move this HSRP address from the unavailable router to the now active router by deleting the HSRP interface from the first router and moving it to the second router.

Ensure that the system.sysName MIB variable is different for each router that is being monitored for the same HSRP IP address.

Netmon provides two HSRP events IBM_NVHSRPADD and IBM_NVHSRPDEL, which are generated when an HSRP interface is added or deleted from an HSRP router so that the network administrator can take the appropriate action. When an HSRP IP address is no longer being used, netmon removes the HSRP IP address when netmon does a configuration check of the router that has the HSRP interface.

Use the -S option for Secondary interfaces to discover and support HSRP virtual interfaces. When an HSRP virtual interface is discovered it will be polled at the status poll interval to detect when it becomes active on another router. You can also create entries in the netmon seed file for the HSRP virtual interface. This is necessary when the HSRP virtual interface in the router has been configured to use the burned in physical address instead of the HSRP specific physical address.
The netmon daemon uses ICMP Echo requests to maintain the status of its managed nodes. It also uses the SNMP to poll the managed nodes for information. For nodes that do not support SNMP, the netmon daemon also uses ICMP Network Mask requests to monitor additional configuration information. Various parameters of the polling operations, including SNMP community names, status polling intervals, and initial time out values, are configured through the Options..SNMP menu item.

Certain interfaces can not respond to ICMP pings, for example unnumbered serial interfaces. Netmon can use SNMP to poll for status by checking the ifAdminStatus and ifOperStatus in the MIBII Interfaces table. If the administration status is not UP, the status for the interface is set to USER1 and will be ignored for status propagation. If the administration status is UP, the operational status is used. If the operational status is not UP, it is deemed DOWN.

You can create entries in the netmon seed file for any node that supports SNMP to activate SNMP for status instead of pings. In addition, netmon will automatically configure routers containing unnumbered serial links for SNMP status. You can also add the P switch to the \usr\ov\conf\oid_to_type file for classes of devices to specify that SNMP be used for status instead of pings for these devices.

If a node is configured for SNMP status, demand poll will use SNMP instead of pings for the status check. This can be overridden using the -E switch. SNMP status polling is controlled from the node level. If a node is configured for SNMP status, it will be polled every polling interval unless the node is unmanaged. Interface status has no affect.

Netmon verifies and sets the status for the node or interface when it receives the following unsolicited, generic traps: Warm Start, Cold Start, Link Down, and Link Up.

The netmon daemon results are displayed on the NetView submaps. The trapd daemon is used to pass messages between NetView and the netmon daemon.

The related graphical user interface operation is Options..Polling; then use the Polling Options dialog box.

The netmon daemon can be configured through the NetView setup options. Set options for NetView daemons on the NetView Options submenu.

Flags

-\texttt{c} Ignores source route bit in the physical address. Use this flag if you are using Cabletron hubs.

-\texttt{J} Causes netmon to attempt to speed the process of discovering new nodes at the expense of limited broadcast traffic. When a new network or subnet is first discovered, netmon causes the first sufficiently capable node it discovers in that network or subnet to broadcast an ICMP Echo request. Thereafter, while the network or subnet remains in the map, netmon causes no additional broadcasts to be generated there.

For the -J option to be effective for a given network or subnet, netmon must discover an SNMP agent that can be instructed to send an ICMP echo request to a given IP address.

This option should be used in environments where very few nodes support SNMP or where the use of Proxy ARP is prevalent. While its use may permit
nodes to be found sooner, in general no nodes will be discovered that would not have been found in the absence of the option.

-K value
Deactivates the Router Fault Isolation feature. This feature is turned on by default. If the Router Fault Isolation feature is not desired, this option can be used to turn it off.

This option has the following syntax:
- -K 0 Turn Router Fault Isolation OFF
- -K 1 Turn Router Fault Isolation ON. This is the default setting.

-k value
Controls the status polling behavior for nodes and routers with a status of Unreachable. By default, netmon will suppress pings and all SNMP requests to nodes in Unreachable networks, but will status poll routers in Unreachable networks. This option can be used to change the behavior, to either suppress all polling to Unreachable routers or to turn on polling of nodes (non-routers) in Unreachable networks.

This option has the following syntax:
- -k 0 Do not suppress any pings or SNMP requests.
- -k 1 Suppress pings and all SNMP requests to non-routers in Unreachable networks. This is the default.
- -k 2 Suppress pings and all SNMP requests to Unreachable routers. If this option is used, it may be necessary to manually ping a node in an occluded area to start recovery for some isolated routers.

-I
Discovers networks and segments that are created as a result of the loadhosts command adding nodes as managed. The default is to discover the networks and segments as unmanaged. Creates new networks as managed.

Note: If this option is used, it may be necessary to manually ping a node in an occluded area to start recovery for some isolation routers.

-m tracemask
Sets the initial trace mask to tracemask. The default is No Tracing. See "Trace Masks" on page 63 for valid trace mask values.

-M tracemask
Sends a message to the currently executing netmon to change its trace mask to tracemask. See "Trace Masks" on page 63 for valid trace mask values.

-O objidfile
Uses the file objidfile instead of the \usr\ov\conf\oid_to_type file to help select the default icon for a node. To construct the objidfile, use the format described at the beginning of the \usr\ov\conf\oid_to_type file. If you incorrectly specify the object type for an object ID, the results will be unpredictable.

-P
Indicates to netmon that it was started by ovspmd, the NetView process management daemon. The netmon daemon will coordinate and communicate with the ovspmd daemon as a member of the OVs_WELL_BEHAVED class. This option should only be used in the local registration file (LRF) file for netmon, \usr\ov\lrfnetmon.lrf.

-s seedfile
Uses the nodes listed in the file seedfile for polling and sets the initial management region to the networks in which those nodes are contained.
The nodes in the seedfile must be IP addresses or hostnames, and they should support SNMP.

The seedfile must have one entry per line. A # character indicates the beginning of a comment. Characters from the # character to the end of the line are ignored.

Routers that support SNMP are the best candidates for inclusion in the seedfile. If the network topology database already exists when the netmon daemon is invoked, netmon ignores nodes in the seedfile that are already in the topology database.

If seedfile is not specified and the topology database does not exist, a map is created using the management node (the node that is executing the netmon daemon).

-t tracefile
Uses the file tracefile as the trace output file, rather than the default trace output file \usr\ov\log\netmon.trace. See "Trace Masks" on page 63 for valid trace mask values.

-S Discovers secondary addresses for devices, such as routers or gateways, that support secondary addressing. This option is set by default and enables support for HSRP.

Using the -S option increases network traffic.

-S Search node ARP caches for HSRP addresses. Using this will speed up HSRP discovery and not require the HSRP virtual address to be explicitly listed in the seed file, (%192.168.1.1)

Note: Using this option in an environment where HSRP routers run with proxy ARP enabled, may degrade performance. A node that is using proxy ARP for routing information has an ARP cache with most entries resolving to the router's HSRP physical address. This will cause netmon to mistakenly identify the interface as an HSRP interface. It is recommended that users opt to list HSRP interfaces in the netmon seed file and leave off this flag.

-u Supports additional discovery and management of IP nodes that exist in an IP network and are connected through an interface that does not have an explicit IP address (unnumbered serial links).

-Q number
Sets the queue size for SNMP pings. The value you specify with this switch can degrade system performance and response time, therefore, we recommend that you use Server Setup to regulate netmon behavior.

-q number
Sets the queue size for ICMP pings. The value you specify with this switch can degrade system performance and response time, therefore, we recommend that you use Server Setup to regulate netmon behavior.

-g number
Specifies the number of hours to elapse before down DHCP nodes are deleted. The default is 12 hours. The valid range is 1 ...999.

-z value
Specifies the status to be assigned to DHCP client nodes and interface cards that are down

This option has the following syntax:
-z 0 Set the status to User1. This is the default.
- z 1 Set the status to Critical.

-f Causes netmon to use Ping Spray during initial discovery only. The values you specify with this switch can degrade system performance and response time, therefore, we recommend that you use Server Setup to regulate netmon behavior.

-F Adjusts the netmon -Q variable for a very fast initial discovery, but at the expense of slower response time. The -F switch impacts the -Q switch. If you specify the -F switch and the network has not been discovered yet, then -Q is automatically adjusted to a queue size of 32 during initial discovery, and re-adjusted to a queue size of 6 after initial discovery (during a steady state). The values you specify with this switch can degrade system performance and response time, therefore, we recommend that you use Server Setup to regulate netmon behavior.

-h Specify this option to use alternate community strings during all SNMP polls.

When the default name public fails, netmon will automatically try the community strings defined in \usr\OV\conf\communityNames.conf under the following circumstances:
- Initial discovery for all nodes.
- Configuration polling and New Node Discovery polling of routers only.
- Demand polling for any node.

When a string is found that works, it will be stored for future use in the ovsnmp.conf database.

This option should be used judiciously because it will slow down polling noticeably due to timeouts for non-SNMP nodes. Ideally, you would want to use this option when a large number of community strings have been changed. It is intended to be used during a full configuration cycle (which is usually 24 hours), and then turned off again.

-E By default netmon will not perform PING requests during a demand poll on devices which are configured to use SNMP Status for polling. Add this switch to force netmon to always perform PING requests as well during a demand poll on these devices.

Trace Masks: Trace masks specify the type of output listed in \usr\OV\log\netmon.trace or in tracefile. To select multiple output types, add the individual tracemask values together and enter that number. Use the following trace mask values:

<table>
<thead>
<tr>
<th>Mask Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off tracing</td>
</tr>
<tr>
<td>1</td>
<td>Trace ICMP Echo requests</td>
</tr>
<tr>
<td>2</td>
<td>Trace ICMP Echo replies and time outs</td>
</tr>
<tr>
<td>4</td>
<td>Trace SNMP requests</td>
</tr>
<tr>
<td>8</td>
<td>Trace SNMP replies and time outs</td>
</tr>
<tr>
<td>16</td>
<td>Trace traps generated</td>
</tr>
<tr>
<td>32</td>
<td>Trace traps received</td>
</tr>
</tbody>
</table>

Examples

You can trace ICMP Echo replies and time-outs by entering the following syntax:

netmon -M 2
You can trace SNMP replies and time-outs by entering the following syntax:
```
netmon -M 8
```

You can trace both ICMP Echo and SNMP replies and time-outs by entering the following syntax, using the sum of their tracemask values (2 + 8) as the tracemask value:
```
netmon -M 10
```

**Implementation Specifics**

The netmon daemon supports single-byte character code sets.

You can configure and control the daemon’s options through NetView setup options.

**Files**

- `\usr\ov\conf\ovsnmp.conf`
  - Community names for SNMP request

- `\usr\ov\log\netmon.trace`
  - Default trace file

- `\usr\ov\conf\oid_to_type`
  - Default object ID file

- `\usr\ov\databases\openview\topo\*`
  - Topology database

- `\usr\ov\databases\openview\topo\netmon.lock`
  - Lockfile for netmon

- `\usr\ov\tmp\demand.*`
  - Temporary files for nmdemandpoll command

- `\usr\ov\lrf\netmon.lrf`
  - Local registration file for the netmon daemon

**Related Information**

- See "loadhosts" on page 44.
- See "lrf" on page 48.
- See "oid_to_type" on page 273.
- See "nmdemandpoll" on page 69.
- See "ovsnmp.conf" on page 304.
- See "trapd" on page 700.
netview

Purpose

Executes the NetView graphical user interface.

Syntax

```
netview [-ro|-rw] [-map map_name ]
```

Description

You can use the `netview` command to start the NetView program. The netview command supports the same options as those the ovw command supports.

Flags

In addition to most standard command-line options, the netview command supports the following options:

- `-map map_name`
  Opens map `map_name`. Creates the map `map_name` if it does not exist. The default behavior is to open the user's default map based on the user's UID. If no user default map is configured, the graphical user interface opens the map named default.

- `-ro`
  Opens all maps as read-only. This option may not be specified at the same time as the `-rw` option.

- `-rw`
  Opens all maps as read-write if possible. This option is the default behavior.

Libraries

If the \usr\ov\bin directory is not in your PATH when you are using the netview command, add it to your PATH or execute the \usr\ov\bin\netview command path.

<table>
<thead>
<tr>
<th>Object Purpose</th>
<th>OVwDB Field Name</th>
<th>OVw Field Type</th>
<th>Field Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global capabilities</td>
<td>“vendor”</td>
<td>ovwEnumField</td>
<td>Capability General Locate</td>
</tr>
<tr>
<td>Global name fields</td>
<td>“SelectionName”</td>
<td>ovwStringField</td>
<td>Name Loc</td>
</tr>
<tr>
<td></td>
<td>“IP Hostname”</td>
<td>ovwStringField</td>
<td>Name Locate</td>
</tr>
<tr>
<td>Location capabilities</td>
<td>“isNetwork”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isInternet”</td>
<td>ovwBooleanField</td>
<td></td>
</tr>
<tr>
<td>Segment capabilities</td>
<td>“isSegment”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isBusSegment”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isStarSegment”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isTokenRingSegment”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isFDDIRingSegment”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>Nodal capabilities</td>
<td>“isNode”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td></td>
<td>“isDevice”</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
</tbody>
</table>
### Table 9. NetView Registration Fields (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;isComputer&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isConnector&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isBridge&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isRouter&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isHup&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isRepeater&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Computer types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isPC&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isWorkstation&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isMini&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isMainFrame&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Device types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isPrinter&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isAnalyzer&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Connection capabilities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isConnection&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Interface capabilities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isCard&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isInterface&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isStarInterface&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isThinInterface&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isThickInterface&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isSerialInterface&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Client/server capabilities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isClient&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td>&quot;isServer&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>Software Capabilities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isSoftware&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
<tr>
<td><strong>IP Specific Capabilities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;isIP&quot;</td>
<td>ovwBooleanField</td>
<td>Capability</td>
</tr>
</tbody>
</table>

### Libraries

When compiling a program that uses the EUI API, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

### Files

- `\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\reg\beginner\*`
- `\usr\ov\reg\advanced\*`
Related Information

- See \texttt{ovstart} on page 358.
- See \texttt{ovwdb} on page 464.
- See \texttt{OVwRegintro} on page 591.
- See the \textit{User's Guide}.
netviewd

Purpose

This is the daemon version of the NetView Console. It runs unattended and serves as a map server to the Web Clients.

Syntax

netviewd -spmd

Description

The netviewd daemon is a special version of the NetView Console that is able to run at daemon startup without a user having to be logged in to a native console. The netviewd daemon performs the function of supplying the network map for the Web Clients without a user needing to log onto the Tivoli NetView server and starting up the regular NetView console. It can only be started by ovspmd and is registered with ovspmd using either the ovaddobj command or through the Server Setup application's Daemons→Web Server→Console Daemon menu option.

The netviewd daemon can not run at the same time as the regular NetView Console. The state of the netviewd daemon is monitored by the nmpollingd daemon.

Flags

- spmd
  Required. Causes netviewd to connect to ovspmd.

Files

\usr\ov\lrf\netviewd.lrf
  The LRF file.
\usr\ov\bin\netviewd
  The command path.

Related Information

- See "netview" on page 65.
- See "polld" on page 671.
nmdemandpoll

Purpose
Polls a node for information.

Syntax
nmdemandpoll nodename ...

Description
The nmdemandpoll command sends a request to the netmon daemon for it to poll the specified node(s) in sequence. The nodename must exist in the NetView object database; the nodename corresponds to the Object Name of the object in the database. If the Domain Name Service is in use, the Object Name is usually the fully-qualified domain name of the device.

The nmdemandpoll polls the following information in the sequence listed: status, configuration, and new node discovery. The output contains the results of these polls. The polling checks occur regardless of the currently set polling switches and intervals in the graphical interface.

The graphical user interface equivalent to nmdemandpoll is the Test..Demand Poll operation.

Implementation Specifics
Only one nmdemandpoll can be executed for a given node at a time.

The netmon daemon limits the number of simultaneous polling operations; therefore, nmdemandpoll may have to wait until outstanding requests are completed.

Files
/usr/ov/tmp/demand.*
Temporary files for communicating with the netmon daemon.

Related Information
- See netview on page 63
- See netmon on page 59
nmpolling

Purpose

Provides topology and status polling configurations.

Syntax

nmpolling

Description

If the NetView daemons on the NetView server stop for any reason, you will not be able to restart them from a remote NetView client. If your NetView server is not readily available to your client(s), it is recommended that you run the nmpolling application by selecting (Options→Polling from the NetView console) on the NetView server with the daemon Auto-start checkboxes enabled. The nmpolling application automatically runs if the NetView GUI is running on the server. However, if the NetView GUI is not running on the server, you can still run the nmpolling application by simply entering the nmpolling command at the command prompt on the NetView server, then select the Daemons tab page and check the desired Auto-restart checkboxes.

The nmpolling application is a NetView graphical user interface application that enables users to configure certain netmon polling intervals. The nmpolling application also monitors the status of all NetView daemons (basic daemons, event daemons, snmpCollect daemon, and the open topology daemon) and can restart the daemon groups by default.

The nmpolling application registers in the graphical user interface menu bar under the Options..Polling menu item. When this item is selected, nmpolling presents a dialog box that contains two tab pages: Nodes and Daemons. The Nodes tab page enables you to do the following:

- Turn netmon polling on or off.
- Configure the interval for netmon to delete nodes that are down.
- Turn netmon discovery on or off.
- Configure netmon for auto-adjusting or fixed polling intervals.
- Configure the netmon configuration polling interval.

The polling intervals may be modified only by users with write permission for the directory /usr/ov/databases/openview/topo. The Daemons tab page enables you to do the following:

- Monitor the status of daemon groups and restart daemon groups.
- Specify the interval for checking daemons.
- Automatically restart daemons.

The nmpolling application runs continuously during a graphical user interface session to monitor the status of the NetView daemons. At the specified intervals, nmpolling checks to see if the daemons are running. If a daemon is not running and the Auto-restart box is checked, the daemon is automatically restarted; otherwise, if a daemon is not running, a warning dialog box is displayed.

To disable daemon monitoring or automatic restart, clear the check mark next to the daemon group you do not want monitored and next to Auto-restart. For more information about this tab page, press the help button in this dialog box. See also netmon on page 59.
The npmolling command line options can be set in the npmolling registration file, \usr\ov\reg\advanced\c\npmolling and \usr\ov\reg\beginner\c\npmolling (if you have installed a non–English version of NetView, replace the "c" with the appropriate locale specifier). To change them, edit the Command statement in the registration file.

Files

\usr\ov\reg\advanced\c\npmolling
NetView graphical user interface application registration file

\usr\ov\reg\beginner\c\npmolling
NetView graphical user interface application registration file

Related Information

- See "netmon" on page 58.
- See "netview" on page 65.
- See "ovtopmd" on page 363.
- See "snmpCollect" on page 680.
nvcold

Purpose

The SmartSet facility daemon.

Syntax

nvcold

Description

The nvcold daemon maintains SmartSets as they have been defined by users or applications. It provides a notification method that enables applications to know when a SmartSet has been added, deleted, or modified.

The nvcold daemon is started by default when you issue the ovstart command. During initialization, the nvcold daemon reviews all the objects in the object database to determine whether they should be placed in any defined SmartSets. When running the NetView console, a launches process, collmap, communicates with nvcold to create the SmartSets symbol in the Root submap as well as all symbols and submaps that appear beneath this top-level symbol. As ovwdb objects change field values, SmartSet memberships may change within nvcold and these changes are reflected in the NetView console by collmap.

The initial set of SmartSets that NetView starts with is defined within the \usr\ov\bin\createsmartsets.bat file. In addition, service SmartSets may be created whenever nvsniffer runs. By default, NetView schedules nvsniffer to periodically run discovery and status passes for the service SmartSets defined by \usr\ov\conf\nvsniffer.conf.

User-defined SmartSets can be created within the NetView console by issuing smartsetutil commands.

When recreating databases, smartsetmaint is used if you elect to preserve your current set of SmartSets. Otherwise, if you elect to not preserve your SmartSets when recreating your databases, the default set of SmartSets is read from the \usr\ov\bin\createsmartsets.bat file.

Related Information

See [smartsetmaint on page 677](#).

See [smartsetutil on page 678](#).

See [nvsniffer on page 263](#).

See [nvcoll on page 73](#).
nvcoll

Purpose

Syntax

    nvcoll

Description

Related Information

See "smartsetmaint" on page 677.

See "smartsetutil" on page 678.

See "nvsniffer" on page 263.
nvecho

Purpose
Provides visual user notification for an error.

Syntax
nvecho [message]

Description
The nvecho command provides visual user notification when a user-defined error occurs.

Use this command within a batch file to be run when a specified error, such as an event, occurs. An error dialog box notifies the operator of the error.

The nvecho command consists of the keyword, nvecho, followed by a message. The message is displayed in the newly created error window. If you want the same function with an audible beep, use nvbeep.

Implementation Specifics
The nvecho command supports single-byte character code sets.
nvmail

Purpose

Sends a mail message.

Syntax

```
nvmail -server servername -recipient user-mailbox [-subject subject-string ][-message message-string] [file file-name]
```

Description

The nvmail command can be used to send an e-mail message. It can be invoked upon receipt of a trap to send an e-mail message to the operator in response to the event. Use Options \rightarrow Trap Settings to configure a trap to execute this command.

Flags

- **-server servername**
  Specifies the name of the SMTP mail server.

- **-recipient user-mailbox**
  Specifies the name of the recipient of the mail. Only one recipient may be specified.

- **-subject subject-string**
  A quoted string that will appear as the body of the mail message. This flag is optional

- **-message message-string**
  A quoted string that will appear as they body of the mail message. This option can not be specified if the **-file** option is used.

- **-file file-name**
  A fully qualified file name to be transmitted as the body of the mail message. This option cannot be specified if the **-message** option is used.

Examples

```
nvmail -server mailserver.acme.com -recipient reed@adam12.com -subject "Segment down" -message "Respond as backup"
```

Implementation Specifics

- nvmail will only work with SMTP mail servers.
- Only one recipient can be specified with the nvmail command.
- nvmail does not transmit files as attachments.
nvpage

Purpose

Submits a page to the system-wide pager request queue.

Syntax

```
nvpage [-h host] {-s} | { [-p priority] <pager # | [-n]ID@Carrier> message [message...] }
n```

Description

The nvpage command can be used to submit a page to the system-wide pager request queue. The system wide pager request queue is maintained by the nvpagerd daemon. Nvpagerd is run on the system that has telephony hardware installed and the Windows telephony settings have been established to use the telephony hardware.

You can configure an event to issue a call to a pager when the event is received. To do this, select Options -> Trap Settings and enter the `nvpage` command as an executable command in the Trap Settings dialog.

Flags

```
pager #
  Specifies a direct dial pager number.

--n  Uses a paging carrier’s touch-tone phone number for numeric paging. Only numeric messages should be sent when using this option.

ID@carrier
  Specifies a pager ID number serviced by the specified carrier.

message
  Specifies one or more words separated by spaces or a single word numeric message if you are sending to a numeric pager.

--h host
  Specifies the host that the paging server is on.

--s  Displays a short summary of queued paging messages.

--p priority
  Specifies the queue priority for the message. The range is 0–5, with 0 being the highest priority. The default is 3.
```

Implementation Specifics

The `nvpage` command can be executed on a system in which the pager request daemon is not running as long as the name of the system on which the pager request daemon running is supplied through the `-h` option.

For example, to send a page to an alphanumeric pager 1116676 receiving page through the carrier Caltel in an enviroment where the datamodem is installed on a system named fred.mynet.com, use command:

```
nvpage -h fred.mynet.com 1116676@caltel Meet at 8:00am in the lobby
```
The following command sends the numeric message 4085551234 to the numeric pager that is identified by the number 5558899 through dialing Caltef's 800 number:

```
nvpage -n 5558899@caltef 4085551234
```

The following command sends the numeric message 4085551234 to the numeric pager that can be dialed directly with the number 5556789:

```
nvpage 5556789 4085551234
```
NVgetServerName

Purpose
Determines the name of the server to which a client should connect.

Syntax

```c
#include <nv_client_server.h>

void NVgetServerName (char * ServerName, int nameLen)
```

Description

This function determines the name of the server that a client application should connect to in a client/server environment.

When you use this routine on the client, it returns the server name stored in NetBIOS. When you use this routine on the server, it returns the fully qualified IP name of the local machine. When you use this routine on a single user mode system, it returns the loopback name localhost.

The 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, NVgetServerName returns the contents of the file. If the file does not exist, the machine is a server, and NVgetServerName returns the current host name.

Return Values

If successful, NVgetServerName returns a server name; otherwise, the string is zero (0).

Libraries

When compiling a program that uses NVgetServerName, link to the \usr\pv\lib\libov.lib import library.

Files

When compiling a program that uses NVgetServerName, include the \usr\ov\include\nv_client_server.h file.

Related Information

See "NVisClient" on page 80
NVinstallMode

Purpose

Returns the installation mode of the NetView program: client, server, or single user mode.

Syntax

#include <nv_client_server.h>

int NVinstallMode()

Description

The NetView program can be installed as a client, server, or single user mode. Clients can connect to server installations, but cannot connect single user mode installations.

Return Values

NVinstallMode returns CLIENT_MODE, SERVER_MODE, or SINGLE_MODE. These macros are defined in nv_client_server.h.

Libraries

When compiling a program that uses NVinstallMode, link to the \usr\ov\lib\libov.lib import library.

Files

When compiling a program that uses NVinstallMode, include the \usr\ov\include\nv_client_server.h file.

Related Information

See "NVisClient on page 80."
NVisClient

Purpose
Checks to see if an application is running on a client or a server.

Syntax

```c
#include <nv_client_server.h>

int NVisClient ();
```

Description
The NVisClient function determines whether an application is running on a NetView 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 or a single user installation, NVisClient returns a value of 0 (FALSE). If the machine is a client, NVisClient returns a value of 1 (TRUE).

Libraries
When compiling a program that uses NVisClient, link to the \usr\ov\lib\libov.lib import library.

Files
When compiling a program that uses NVisClient, include the \usr\ov\include\nv_client_server.h file.

Related Information
See "NVgetServerName" on page 78.
nvotChangeArcDetails

Purpose

Changes the contents of the details variable in the database.

Syntax

```c
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 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.

Parameters

```c
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" on page 250. Regardless of which direction was set in the nvotInit routine, the arcNameBinding parameter 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, must take 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 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_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_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. Try
reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg
routine, as shown in the following example:

```c
nvotReturnCode rc;

if (((rc = nvotGetError()) != NVOT_SUCCESS))
    printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the character string \texttt{mystring} in the \texttt{myLineArcDetails}
variable of the arc created in the example in \href{nvotCreateArcInGraph}{nvotCreateArcInGraph} on page 158.
```c
#include <nvot.h>

nvotReturnCode rc;
char myString[128] = "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;

myLineArcDetails.octetString = malloc(sizeof(myString));
memcpy(myLineArcDetails.octetString, myString, sizeof(myString));

if ((rc = nvotChangeArcDetails (ARC_VERTEX_VERTEX_NAME_BINDING,
oneEndpointType, 
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**

When compiling a program that uses `nvotChangeArcDetails`, link to the `/usr/OV/lib/libnvot.lib` library.

**Files**

`nvot.h`

**Related Information**

- See [nvotCreateArcInGraph](#) on page 158
nvotChangeArcIconInGraph

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, aEndpointName, 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 /usr/OV/conf/oid_to_protocol file.

**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** on page 250. Regardless of the selection made in the nvotInit routine, arcNameBinding 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, must take 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.

arclIndexId
Distinguishes an arc from other arcs between the same endpoints. (Two endpoints can be connected by several different arcs.) The arclIndexId is an integer value.

icon
Specifies a new symbol to represent the arc in the NetView EUI. 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 158.

```c
#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 * myDashArcIcon = "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,
    myDashArcIcon)) == 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

When compiling a program that uses nvotChangeArcIconInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotChangeArcLabelInGraph" on page 89
- See "nvotCreateArcInGraph" on page 158
nvotChangeArcLabelInGraph

Purpose

Changes an arc label in a graph.

Syntax

```c
#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`, `aEndpointName`, `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 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 /usr/OV/conf/oid_to_protocol file.

**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” on page 250. 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, must take 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 158.

```c
#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
When compiling a program that uses nvotChangeArcLabelInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files
nvot.h
Related Information

- See `nvotChangeArcIconInGraph` on page 85.
- See `nvotCreateArcInGraph` on page 158.
nvotChangeArcStatus

Purpose

Changes the status of an arc.

Syntax

```
#include <nvot.h>

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. If the value passed is not valid, the operation is rejected and error code `NVOT_INVALID_STATUS` is returned. For more information refer to the `Tivoli NetView for Windows 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" on page 250. 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, must take 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. When 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 158.

```c
#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", nvotGetErrorMsg (rc));

**Libraries**

When compiling a program that uses nvotChangeArcStatus, link to the /usr/OV/lib/libnvot.lib library.

**Files**

nvot.h

**Related Information**

- See [nvotCreateArcInGraph](#) on page 158
- See [nvotGetArCSInGraph](#) on page 228
- See [nvotInit](#) on page 250
nvotChangeBoxBackground

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. Try reissuing the nvotInit routine.

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" on page 163.
#include <nvot.h>

ntReturnCode rc;

ntGraphProtocolType 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

When compiling a program that uses `nvotChangeBoxBackground`, link to the `/usr/OV/lib/libnvot.lib` library.

## Files

nvot.h

## Related Information

- See [nvotCreateRootGraph](#) on page 178
- See [nvotCreateGraphInGraph](#) on page 169
- See [nvotCreateBoxInGraph](#) on page 164
nvotChangeBoxDetails

Purpose

Changes the contents of the details variable in the general topology manager (GTM) database

Syntax

```c
#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 `/usr/OV/conf/oid_to_protocol` file.

**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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

if (rc != NVOT_SUCCESS)
    printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example stores the contents of myStruct and myString into the boxDetails variable attribute of My_Box:

```c
#include <nvot.h>
typedef struct { int var1;
    int var2;
 } structType;

structType myStruct = { 11, 22 }; char myString [128] = "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;
```
Libraries

When compiling a program that uses nvotChangeBoxDetails, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateRootGraph" on page 178
- See "nvotCreateBoxInGraph" on page 164
nvotChangeBoxIconInGraph

Purpose
Changes a box graph icon in a graph.

Syntax

```c
#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 /usr/OV/conf/oid_to_protocol file.

`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 /usr/OV/conf/oid_to_protocol file.

`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.
Specifies a symbol to represent the child box graph in the NetView 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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 164.
#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

When compiling a program that uses nvotChangeBoxIconInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateBoxInGraph" on page 164
- See "nvotChangeBoxLabelInGraph" on page 107
nvotChangeBoxLabelInGraph

Purpose

Changes a box graph label in a graph.

Syntax

```c
#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 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 /usr/OV/conf/oid_to_protocol file.

- `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 /usr/OV/conf/oid_to_protocol file.

- `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 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**

```c
[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. Try reissuing the `nvotInit` routine.
```

A printable message string is accessible through a call to the `nvotGetErrorMsg` routine, as shown in the following example:

```c
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" on page 164.
#include <nvot.h>

```c
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

When compiling a program that uses nvotChangeBoxLabelInGraph, link to the /usr/OV/lib/libnvot.lib library.

## Files

nvot.h

## Related Information

- See [nvotCreateBoxInGraph](#) on page 164
- See [nvotChangeBoxIconInGraph](#) on page 104
nvotChangeBoxPositionInGraph

Purpose
Changes the position of a box graph icon in a graph submap.

Syntax
```c
#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 /usr/OV/conf/oid_to_protocol file.

`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 /usr/OV/conf/oid_to_protocol file.

`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`
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 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. Try reissuing the nvotInit routine.
A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

if (rc != NVOT_SUCCESS)
    printf("%s\n", nvotGetErrorMsg(rc));
```

**Examples**

The following example changes the position of a child box graph.

```c
#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**

When compiling a program that uses nvotChangeBoxPositionInGraph, link to the /usr/OV/lib/libnvot.lib library.

**Files**

nvot.h

**Related Information**

- See [nvotCreateBoxInGraph](#) on page 164
- See [nvotChangeGraphPositionInGraph](#) on page 125
nvotChangeGraphBackground

Purpose
Changes the background of a graph map.

Syntax

```c
#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_ERRORALLOCATING_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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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](#) on page 163.
#include <nvot.h>

nvotReturnCode rc;

nvotGraphProtocolType mySDLCGraphsProt = "1.3.6.1.2.1.2.1.3.17";
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 is changed.\n", myChildSDLCGraphName);
else
    printf ("Error occurred changing background.\n", myChildSDLCGraphName);
    printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
nvotChangeGraphDetails

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 /usr/OV/conf/oid_to_protocol file.
- **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 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. The 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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.

```c
#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

When compiling a program that uses nvotChangeGraphDetails, link to the  
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotCreateRootGraph” on page 178](#)
- See [nvotCreateGraphInGraph” on page 169](#)
nvotChangeGraphIconInGraph

Purpose

Changes a graph icon in a graph.

Syntax

```c
#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 `/usr/OV/conf/oid_to_protocol` file.

- **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 `/usr/OV/conf/oid_to_protocol` file.

- **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 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. The 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. Try
    reissuing the nvotInit routine.

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" on page 169

#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 = "1.3.6.1.2.1.2.1.3.10.11";
if (rc = nvotChangeGraphIconInGraph (mySDLGGraphsProt,
    myRootSDLGGraphName,
    mySDLGGraphsProt,
    myChildSDLGGraphName,
    mySDLGGraphIcon) == NVOT_SUCCESS) {
    printf ("Graph icon of graph %s changed.\n", myChildSDLGGraphName);
} else {
    printf ("An error occurred changing %s icon.\n", myChildSDLGGraphName);
    printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
}

Libraries

When compiling a program that uses nvotChangeGraphIconInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateGraphInGraph" on page 168.
- See "nvotChangeGraphLabelInGraph" on page 122.
nvotChangeGraphLabelInGraph

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 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 /usr/OV/conf/oid_to_protocol file.

- **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 /usr/OV/conf/oid_to_protocol file.

- **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 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. Try reissuing the `nvotInit` routine.

A printable message string is accessible through a call to the `nvotGetErrorMsg` routine, as shown in the following example:

```c
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" on page 169.

```c
#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));

#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

When compiling a program that uses nvotChangeGraphLabelInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateGraphInGraph" on page 169
- See "nvotChangeGraphIconInGraph" on page 119.
nvotChangeGraphPositionInGraph

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. Try reissuing the nvotInit routine.

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 mySDLGraphsProt = "1.3.6.1.2.1.2.2.1.3.17";

char * myRootSDLGraphName = "My Root Graph";
char * myChildSDLGraphName = "My Child SDL Graph";

nvotPositionType myGraphPosition = { 500, /* xCoordinate */
                                      500, /* yCoordinate */
                                      1000, /* xGrid */
                                      1000 /* yGrid */
                                  };

if ((rc = nvotChangeGraphPositionInGraph (mySDLGraphsProt,
                                          myRootSDLGraphName,
                                          mySDLGraphsProt,
                                          myChildSDLGraphName,
                                          myGraphPosition)) == NVOT_SUCCESS)

    printf ("Positioning of graph %s symbol changed.\n", myChildSDLGraphName);
else
    printf ("An error occurred changing %s icon position.\n", myChildSDLGraphName);

printf ("Operation result : %s\n", nvotGetErrorMsg (rc));

Libraries

When compiling a program that uses nvotChangeGraphPositionInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotCreateGraphInGraph](#) on page 169.
- See [nvotChangeBoxPositionInGraph](#) on page 110.
nvotChangeRootGraphIcon

Purpose
Changes a root graph icon.

Syntax

```c
#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 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 NVOT_ROOT_GRAPH_DOES_NOT_EXIST error code 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 /usr/OV/conf/oid_to_protocol file.

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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

If (rc != NVOT_SUCCESS)
    printf ("%s
", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the root graph created in the example in "nvotCreateRootGraph" on page 178.


```c
#include <nvot.h>

nvotReturnCode rc;

nvotGraphProtocolType myRootGraphProt = "1.3.6.1.2.1.2.1.3.17";
char * myRootGraphName = "My Root Graph";
char * my_NEW_RootGraphIcon = "1.3.6.1.2.1.2.1.3.9.11";

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**

When compiling a program that uses `nvotChangeRootGraphIcon`, link to the `/usr/OV/lib/libnvot.lib` library.

**Files**

nvot.h

**Related Information**

- See [`nvotCreateRootGraph` on page 178](#).
- See [`nvotChangeRootGraphLabel` on page 131](#).
nvotChangeRootGraphLabel

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 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 NVOT_GRAPH_INVALID_INDEX error code is returned.

If the root graph specified does not exist in the GTM database, the label does not exist and the NVOT_ROOT_GRAPH_DOES_NOT_EXIST error code 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 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 /usr/OV/conf/oid_to_protocol file.

**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 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_GTM_D_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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

cmpotReturnCode 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" on page 178:
```c
#include <nvot.h>

nvotReturnCode rc;

nvotGraphProtocolType myRootGraphProt = "1.3.6.1.2.1.2.1.3.1.17";
char * myRootGraphName = "My_Root_Graph";
char * my_NEW_RootGraphLabel = "My.NEW_ROOT_GRAPH_LABEL";

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

When compiling a program that uses `nvotChangeRootGraphLabel`, link to the `/usr/OV/lib/libnvot.lib` library.

### Files

`nvot.h`

### Related Information

- See [nvotCreateRootGraph](#) on page 178
- See [nvotChangeRootGraphic](#) on page 128
nvotChangeVertexDetails

Purpose

Changes the contents of the details variable in the GTM database.

Syntax

```c
#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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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:

```c
#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

When compiling a program that uses `nvotChangeVertexDetails`, link to the `/usr/OV/lib/libnvot.lib` library.

Files

nvot.h
Related Information

• See "nvotCreateRootGraph" on page 178.
• See "nvotCreateGraphInGraph" on page 169.
nvotChangeVertexIconInBox

Purpose

Changes a vertex icon in a box.

Syntax

```c
#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 NVOT_BOX_INVALID_INDEX or NVOT_VERTEX_INVALID_INDEX error code is returned.

If the containing box graph does not exist, the vertex icon is not changed and the NVOT_BOX DOES NOT_EXIST error code 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 /usr/OV/conf/oid_to_protocol file.

- **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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 185. Icon is changed to “Cards:Generic”: 

138 Programmer’s Reference
#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

When compiling a program that uses nvotChangeVertexIconInBox, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

• See `nvotCreateVertexInBox` on page 185.
nvotChangeVertexIconInGraph

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. 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 `/usr/OV/conf/oid_to_protocol` file.

- `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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;
If (rc != NVOT_SUCCESS)
    printf ("%s
", nvotGetErrorMsg (rc));
```

Examples

The following example changes the icon of the vertex created in the example in "nvotCreateVertexInGraph" on page 189.

```c
#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));

Related Information

- See "nvotCreateVertexInGraph" on page 188
nvotChangeVertexLabelInBox

Purpose

Changes the label of a vertex in a box.

Syntax

```c
#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 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 /usr/OV/conf/oid_to_protocol file.

- **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.
Specifies a symbol to represent the vertex in the NetView 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

If (rc != NVOT_SUCCESS)
   printf ("%s
", nvotGetErrorMsg (rc));
```

Examples

The following example changes the label of the vertex created in the example in "nvotCreateVertexInGraph" on page 189.
#include <nvot.h>

nvotReturnCode rc;

nvotGraphProtocolType myBoxProt = "1.3.6.1.2.1.2.1.3.11";
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

When compiling a program that uses `nvotChangeVertexLabelInBox`, link to the `/usr/OV/lib/libnvot.lib` library.

### Files

nvot.h

### Related Information

- See [nvotCreateVertexInBox](#) on page 185.
nvotChangeVertexLabelInGraph

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 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. 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

- **graphProtocol**
  Specifies the protocol of the containing graph. For more information about specifying a graph's protocol, refer to the `/usr/OV/conf/oid_to_protocol` file.

- **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 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 189.

```c
#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**

When compiling a program that uses `nvotChangeVertexLabelInGraph`, link to the `/usr/OV/lib/libnvot.lib` library.

**Files**

nvot.h

**Related Information**

- See [nvotCreateVertexInGraph](#) on page 188.
nvotChangeVertexPositionInBox

Purpose

Changes position of a vertex icon in a box submap.

Syntax

```c
#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 zooming out.

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 `/usr/OV/conf/oid_to_protocol` file.

**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_GTMDINVALID_RESPONSE]**
The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

**[NVOT_ERRORALLOCATING_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. Try reissuing the `nvotInit` routine.

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

When compiling a program that uses nvotChangeVertexPositionInBox, link to the 
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateBoxInGraph" on page 164.
- See "nvotChangeVertexPositionInGraph" on page 152.
nvotChangeVertexPositionInGraph

Purpose
Changes position of a vertex icon in a graph submap.

Syntax

```c
#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

\textit{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.

\textit{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.

\textit{vertexProtocol}
Specifies the protocol of the vertex. Vertex protocol is an enumerated type defined in the file nvotTypes.h.

\textit{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.

\textit{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

\textit{nvotReturnCode}
The \textit{nvotchangeVertexPositionInGraph} routine returns an \textit{nvotReturnCode} that can assume the values described in the following error codes section.

Error Codes

\textbf{[NVOT_SUCCESS]}
Successful operation.

\textbf{[NVOT_GRAPH_INVALID_INDEX]}
The graph index is not valid. A graph protocol or name must not be NULL.

\textbf{[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.

\textbf{[NVOT_GRAPH_DOES_NOT_EXIST]}
A graph does not exist in the GTM database.

\textbf{[NVOT_GTMD_INVALID_RESPONSE]}
The GTM response is not valid. A query to a graph or member table returned an unexpected response from gtmd.

\textbf{[NVOT_ERROR_ALLOCATING_MEMORY]}
Memory allocation error. The system might be out of memory.

\textbf{[NVOT_NOT_INITIALIZED]}
Not initialized. Issue the \textit{nvotInit} routine to establish a connection with gtmd.

\textbf{[NVOT_SOCKET_ERROR]}
There is a socket error. The connection went down during operation. Try reissuing the \textit{nvotInit} routine.

A printable message string is accessible through a call to the \textit{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, signal protocol,               
        myVertexName, signal protocol,                
        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

When compiling a program that uses nvotChangeVertexPositionInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateVertexInGraph" on page 189.
- See "nvotChangeVertexPositionInBox" on page 149.
nvotChangeVertexStatus

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 EUI.

The protocol and name parameters uniquely identify the objects in the GTM database. The vertexProtocol and vertexName parameters are required. If one of these parameters is not provided, the NVOT_VERTEX_INVALID_INDEX error code 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, refer to the section about state management variables in the Tivoli NetView for Windows 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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

defineReturnCode 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" on page 189.

#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

When compiling a program that uses nvotChangeVertexStatus, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateVertexInGraph" on page 189.
nvotCreateArcInGraph

Purpose

Creates an arc in a graph.

Syntax

```c
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 state values according to the table shown in the *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 /usr/OV/conf/oid_to_protocol file.

**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. When 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 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 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](#) on page 250. 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 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 object database.

**status**
Specifies the status of the arc. Arc status is an enumerated type defined in the file <nvotTypes.h>. For more information, see the Tivoli NetView for Windows 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 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 immediately returns OVwNullObjectId. In either case, upon return, an error code is available through a call to the routine nvotGetError. See "nvotSetSynchronousCreation" on page 259 for more information about OVwObjectId.

**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_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_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 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. Try reissuing the nvotInit routine.

[NVOT_OVW_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView database might be down. Try increasing the timeout value.

[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 one arc between two vertex points. Before creating the arc, you must do the following:

1. Create a root graph (see the example in nvotCreateRootGraph on page 178).
2. Create two vertices (V1 and V2) inside the graph (see the example in "nvotCreateVertexInGraph" on page 189).

```c
#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**

When compiling a program that uses `nvotCreateArcInGraph`, link to the `/usr/OV/lib/libnvot.lib` library.

**Files**

`nvot.h`

**Related Information**

- See "nvotDeleteArcFromGraph" on page 196.
- See "nvotChangeArcIconInGraph" on page 85.
- See "nvotChangeArcLabelInGraph" on page 89.
• See `nvotGetArcsInGraph` on page 228.
• See `nvotSetSynchronousCreation` on page 253.
• See `nvotInit` on page 250.
nvotCreateBoxInGraph

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 (is not specified) 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 /usr/OV/conf/oid_to_protocol file.

- **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 /usr/OV/conf/oid_to_protocol file.
**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 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 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 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 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 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](#) on page 259.

## 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_GRAPH_INVALID_INDEX]
The graph index is not valid. A graph protocol or name must not be NULL.

[NVOT_BOX_INVALID_INDEX]
The box index is not valid. A box graph protocol 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. Try reissuing the nvotInit routine.

[NVOT_OVW_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView database might be down. Try increasing the timeout value.

[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:

```c
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

When compiling a program that uses nvotCreateBoxInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateBoxInGraph" on page 178
- See "nvotCreateVertexInGraph" on page 189
- See "nvotCreateGraphInGraph" on page 169
- See "nvotDeleteBoxFromGraph" on page 202
- See "nvotChangeBoxIconInGraph" on page 104
- See "nvotChangeBoxLabelInGraph" on page 107
- See "nvotChangeBoxPositionInGraph" on page 110
- See "nvotGetBoxesInGraph" on page 231
- See "nvotSetSynchronousCreation" on page 253
nvotCreateGraphInGraph

Purpose

Creates a graph within 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 (is not specified) 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 /usr/OV/conf/oid_to_protocol file.

- **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 /usr/OV/conf/oid_to_protocol file.
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 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 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 supports any size strings and the interface does not check the size of graphChildDetails, any character exceeding 256 will be truncated by the NetView 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 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" on page 259.

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_GRAPH_INVALID_INDEX]
The graph index is not valid. A graph protocol 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]
A socket error occurred. The connection went down during operation.
        Reissue the nvotInit routine.

[NVOT_OVW_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation
        might not be large enough to complete operation processing, or the
        connection to the NetView database might be down. Increase the timeout
        value.

[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:

```c
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";

 OVwObjectId 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**

When compiling a program that uses nvotCreateGraphInGraph, link to the /usr/OV/lib/libnvot.lib library.
Files

nvot.h

Related Information

- See `nvotCreateRootGraph` on page 176.
- See `nvotCreateVertexInGraph` on page 189.
- See `nvotDeleteGraphFromGraph` on page 207.
- See `nvotChangeGraphIconInGraph` on page 119.
- See `nvotChangeGraphLabelInGraph` on page 122.
- See `nvotChangeGraphPositionInGraph` on page 125.
- See `nvotGetGraphsInGraph` on page 238.
- See `nvotSetSynchronousCreation` on page 259.
nvotCreateProvidingSap

**Purpose**

Creates a SAP of SAP type PROVIDING.

**Syntax**

```c
#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.

A Sap 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
document is not known, use OTHER_PROTOCOL as defined in the file
nvotTypes nvotVertexProtocolType. To avoid breaking the preceding rule
(see "the rule" on page 174) 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.
In another case, is a single BOX graph might contain 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 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 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.

[NVOTVERTEX_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. Try reissuing the nvotInit routine.

[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:

```c
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.

```c
#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**

When compiling a program that uses nvotCreateProvidingSap, link to the `/usr/OV/lib/libnvot.lib` library.

**Files**

- `nvot.h`

**Related Information**

- See [nvotCreateProvidingSap](#) on page 174
- See [nvotSetSynchronousCreation](#) on page 259

---

**Chapter 2. Reference Pages 177**
nvotCreateRootGraph

Purpose

Creates a root graph

Syntax

```c
#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 (is not specified) 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 `/usr/OV/conf/oid_to_protocol` file.
**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 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 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 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 object database.

**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 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](#) on page 259.

### 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.
The graph index is not valid. A graph protocol or name must not be NULL.

Invalid layout. The layout must be a number defined in the nvotTypes.h file.

The status is not valid. It must be a number defined in the nvotTypes.h file.

A graph already exists with the same protocol and name for which this call
is attempting to create a graph, box, or root graph.

A box already exists with the same protocol and name for which this call is
attempting to create a graph, box, or root graph.

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.

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.

Memory allocation error. The system might be out of memory.

Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

There is a socket error. The connection went down during operation. Try
reissuing the nvotInit routine.

NetView timeout. The timeout value passed to nvotSetSynchronousCreation
might not be enough for the complete operation processing, or the
connection to the NetView database might be down. Try increasing the
timeout value.

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:

```c
nvotReturnCode rc;
If ((rc = nvotGetError()) != NVOT_SUCCESS)
printf ("%s\n", nvotGetErrorMsg (rc));
```

The following example creates a graph as root of protocol SDLC and layout
POINT_TO_POINT_RING_LAYOUT.

```c
#include <nvot.h>
...
OvObjectId 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

When compiling a program that uses nvotCreateRootGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotChangeRootGraphLabel](#) on page 131
- See [nvotChangeRootGraphIcon](#) on page 128
- See [nvotSetSynchronousCreation](#) on page 253
nvotCreateUsingSap

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 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. Try reissuing the nvotInit routine.

- **[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:

```c
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.

```c
#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

When compiling a program that uses nvotCreateUsingSap, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateProvidingSap" on page 174
- See "nvotSetSynchronousCreation" on page 253
nvotCreateVertexInBox

Purpose

Creates a vertex in a box.

Syntax

```c
#include <nvot.h>

OVwObjectId ovotCreateVertexInBox ( 
    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 state values according to the table shown in the Tivoli NetView for Windows 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 /usr/OV/conf/oid_to_protocol file.

- **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 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 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 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 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](#) on page 259.

### 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_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_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. Try reissuing the nvotInit routine.

[NVOT_OVW_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView database might be down. Try increasing the timeout value.

[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" on page 164.
#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 = "1.3.6.1.2.1.2.1.3.11.1";
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

When compiling a program that uses nvotCreateVertexInBox, link to the /usr/OV/lib/libnvot.lib library.

## Files

nvot.h

### Related Information

- See [nvotCreateBoxInGraph](#) on page 164
- See [nvotDeleteVertexFromBox](#) on page 213
- See [nvotChangeVertexIconInBox](#) on page 137
- See [nvotChangeVertexLabelInBox](#) on page 143
- See [nvotGetVerticesInBox](#) on page 244
- See [nvotSetSynchronousCreation](#) on page 253
nvotCreateVertexInGraph

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 state values according to the table shown in the Tivoli NetView for Windows 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 /usr/OV/conf/oid_to_protocol file.

- **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 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 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 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 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 immediately return OVwNullObjectId. In either case, upon return, an error code is available through a call to the nvotGetError routine. Refer to “nvotSetSynchronousCreation” on page 258 for more details about OVwObjectId.

---

**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_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_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. Try reissuing the nvotInit routine.

[NVOT_OVV_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView database might be down. Try increasing the timeout value.

[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:

```c
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" on page 169.
#include <nvot.h>

OVwObjectId vertexId;
nvotReturnCode rc;
nvotBooleanType synchMode = FALSE;

nvotGraphProtocolType my_STARLAN_GraphProt = "1.3.6.1.2.1.2.1.3.11.11.11";
char * my_STARLAN_GraphName = "My_STARLAN_Graph";

nvotVertexProtocolType myVertexProt = STARLAN;
char * myVertexName = "My_Vertex";
char * myVertexIcon = "1.3.6.1.2.1.2.1.3.11.11.11";
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

When compiling a program that uses nvotCreateVertexInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateRootGraph" on page 178
- See "nvotCreateGraphInGraph" on page 169
- See "nvotDeleteVertexFromGraph" on page 221
- See "nvotChangeVertexIconInGraph" on page 140
- See "nvotChangeVertexLabelInGraph" on page 146
- See "nvotChangeVertexPositionInGraph" on page 153
- See "nvotGetVerticesInGraph" on page 247
- See "nvotSetSynchronousCreation" on page 253
- See "nvotGetError" on page 234
- See "nvotGetErrorMsg" on page 237
nvotDeleteArc

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 open topology side effects in the *Tivoli NetView for Windows 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](#) on page 250. 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, must take 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**

*nvoReturnCode*
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 was lost during operation. Reissue the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;
if (rc != NVOT_SUCCESS)
    printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example deletes arc number 1 connecting oneEndpoint to otherEndpoint:

```c
#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", arcNumber, oneEndpointName, otherEndpointName);
    printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

When compiling a program that uses nvotDeleteArc, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotCreateArcInGraph](#) on page 158
- See [nvotDeleteArcFromGraph](#) on page 196
- See [nvotGetArcsInGraph](#) on page 228
nvotDeleteArcFromGraph

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 effects. These side effects are described in [nvotDeleteArc on page 193](#).

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] on page 250. 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 zEndpoint Protocol, 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, must take 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 went down during operation. Reissue the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 158:

```c
#include <nvot.h>

nvotReturnCode rc;

nvotGraphProtocolType myGraphProt = "1.3.6.1.2.1.2.2.1.3.11";
char * myGraphName = "My_Graph";

char * oneEndpoint.vertexProtocol = STARLAN;
char * oneEndpointName = "My_Vertex_Endpoint";

char * 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

When compiling a program that uses nvotDeleteArcFromGraph, link to the 
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateArcInGraph" on page 158
- See "nvotGetArcsInGraph" on page 228
nvotDeleteBox

Purpose

Deletes a box.

Syntax

```c
#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 open topology side effects in the *Tivoli NetView for Windows 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 went down during operation. Try
reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg
routine, as shown in the following example:

```c
nvotReturnCode rc;
If (rc != NVOT_SUCCESS)
    printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following sample deletes a box graph:

```c
#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

When compiling a program that uses nvotDeleteBox, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotCreateBoxInGraph](#) on page 164.
- See [nvotDeleteBoxFromGraph](#) on page 202.
- See [nvotGetBoxesInGraph](#) on page 231.
nvotDeleteBoxFromGraph

Purpose
Deletes a box from a graph.

Syntax

```
#include <nvot.h>
	nvotReturnCode nvotDeleteBoxFromGraph (nvtGraphProtocolType graphProtocolParent, char * graphNameParent, nvtGraphProtocolType 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" on page 205.

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 /usr/OV/conf/oid_to_protocol file.

- `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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

if ((rc = nvotGetError()) != NVOT_SUCCESS)
    printf("%s
", nvotGetErrorMsg (rc));
```

Examples

The following sample deletes a box that is a member of the root graph:

```c
#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

When compiling a program that uses nvotDeleteBoxFromGraph, link to the /usr/OV/lib/libnvot.lib library.
Files

nvot.h

Related Information

- See `nvotCreateRootGraph` on page 178
- See `nvotCreateBoxInGraph` on page 164
- See `nvotGetBoxesInGraph` on page 231
nvotDeleteGraph

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 open topology side effects in the 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 or name must not be NULL.

- **[NVOT_ERROR_ALLOCATING_MEMORY]**
  Memory allocation error. The system might be out of memory.
There is a socket error. The connection went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

if (rc != NVOT_SUCCESS)
    printf("%s\n", nvotGetErrorMsg (rc));
```

### Examples

The following example deletes the graph created in \textit{nvotCreateGraphInGraph} on page 163:

```c
#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

When compiling a program that uses nvotDeleteGraph, link to the /usr/OV/lib/libnvot.lib library.

### Files

- nvot.h

### Related Information

- See \textit{nvotCreateGraphInGraph} on page 163
- See \textit{nvotCreateRootGraph} on page 178
- See \textit{nvotDeleteGraphFromGraph} on page 207
- See \textit{nvotGetGraphsInGraph} on page 238
nvotDeleteGraphFromGraph

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" on page 203.

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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 163:

```c
#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

When compiling a program that uses nvotDeleteGraphFromGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h
Related Information

- See `nvotCreateRootGraph` on page 178.
- See `nvotCreateGraphInGraph` on page 169.
- See `nvotGetGraphsInGraph` on page 238.
**nvotDeleteProvidingSap**

**Purpose**

Deletes a SAP of SAP type PROVIDING.

**Syntax**

```c
#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" on page 213.

**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 a 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 went down during operation. Try reissuing the nvotInit routine.

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] on page 174.

```
#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,
    myTCP_IP_Prot,
    myTCP_IP_Name)) == NVOT_SUCCESS)
    printf ("Sap provided to TCP/IP has been deleted.\n\n");
else
    printf ("An error occurred deleting sap.\n\n");
    printf ("Operation result : %s\n", nvotGetErrorMsg (rc));
```

Libraries

When compiling a program that uses nvotDeleteProvidingSap, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h
Related Information

- See `nvotCreateUsingSap` on page 182.
- See `nvotCreateProvidingSap` on page 174.
- See `nvotDeleteProvidingSap` on page 210.
nvotDeleteUsingSap

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 vertexName 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] on page 210 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 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_ERRORALLOCATING_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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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` on page 182:

```c
#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

When compiling a program that uses nvotDeleteUsingSap, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h
Related Information

- See `fnvotCreateUsingSap` on page 182.
- See `fnvotCreateProvidingSap` on page 174.
- See `fnvotDeleteProvidingSap` on page 210.
nvotDeleteVertex

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 open topology side effects in the
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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;
if (rc != NVOT_SUCCESS)
    printf("%s\n", nvotGetErrorMsg (rc));
```

**Examples**

The following example deletes the vertex created in the example in "nvotCreateVertexInGraph" on page 189.

```c
#include <nvot.h>

nvotReturnCode rc;
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**

When compiling a program that uses nvotDeleteVertex, link to the /usr/OV/lib/libnvot.lib library.

**Files**

nvot.h

**Related Information**

- See "nvotCreateVertexInGraph" on page 189.
- See "nvotCreateVertexInBox" on page 185.
- See "nvotDeleteVertexFromGraph" on page 221.
- See "nvotDeleteVertexFromBox" on page 218.
- See "nvotGetVerticesInGraph" on page 247.
- See "nvotGetVerticesInBox" on page 244.
nvotDeleteVertexFromBox

Purpose

Deletes a vertex from a box.

Syntax

```c
#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" on page 216.

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 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 was lost during operation. Reissue the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;
if (rc != NVOT_SUCCESS)
    printf("%s\n", nvotGetErrorMsg (rc));
```

### Examples

The following example deletes the vertex created in "nvotCreateVertexInBox" on page 185.

```c
#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

When compiling a program that uses nvotDeleteVertexFromBox, link to the /usr/OV/lib/libnvot.lib library.

### Files

nvot.h

### Related Information

- See "nvotCreateVertexInBox" on page 185.
- See "nvotChangeVertexIconInBox" on page 137.
- See "nvotChangeVertexLabelInBox" on page 143.
• See `nvotGetVerticesInBox` on page 244.
nvotDeleteVertexFromGraph

Purpose

Deletes a vertex from a graph.

Syntax

```c
#include <nvot.h>

nvotReturnCode nvotDeleteVertexFromGraph (
    nvotGraphProtocolType graphProtocol,
    char * graphName,         // Required
    nvotVertexProtocolType vertexProtocol,
    char * vertexName);       // Required
```

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" on page 216.

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.
The graph index is not valid. A graph protocol and/or name must not be NULL.

The vertex index is not valid. A vertex protocol must be a positive integer and a vertex name must not be NULL.

Memory allocation error. The system might be out of memory.

Not initialized. Issue the nvotInit routine to establish a connection with gtmd.

There is a socket error. The connection went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;
if (rc != NVOT_SUCCESS)
    printf ("%s\n", nvotGetErrorMsg (rc));
```

The following example deletes the vertex created in "nvotCreateVertexInGraph" on page 189.

```c
#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));
```

When compiling a program that uses nvotDeleteVertexFromGraph, link to the /usr/OV/lib/libnvot.lib library.

```
```

Files

```
```
• See `nvotChangeVertexLabelInGraph` on page 146.
• See `nvotChangeVertexPositionInGraph` on page 152.
• See `nvotGetVerticesInGraph` on page 247.
nvotDone

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 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 = nvotDone()) != 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

When compiling a program that uses nvotDone, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotInit" on page 250
- See "nvotGetError" on page 234
- See "nvotGetErrorMsg" on page 237
nvotFree

Purpose

Release memory allocated by routines in the GTM API.

Related Functions

- nvotFreeArc
- nvotFreeArcList
- nvotFreeBox
- nvotFreeBoxList
- nvotFreeGraph
- nvotFreeGraphList
- nvotFreeSap
- nvotFreeSapList
- nvotFreeVertex
- nvotFreeVertexList

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)
```

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, please see "Chapter 3. Using NetView GTM Data Structures" on page 713.

Return Values

```
nvotReturnCode
The routine returns NVOT_SUCCESS.
```
Error Codes

[NVOT_SUCCESS]
The operation was successful.

Libraries

When compiling a program that uses nvotFree, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h
nvotGetArcsInGraph

Purpose

Gets a list of all arcs contained in a graph.

Syntax

```c
#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 deallocated.

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 713.

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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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] on page 153.

```c
#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);
```
When compiling a program that uses nvotGetArcsInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See nvotFreeArc in "nvotFree" on page 226.
- See nvotFreeArcList in "nvotFree" on page 226.
- See "nvotGetError" on page 234.
- See "nvotGetErrorMsg" on page 237.
nvotGetBoxesInGraph

Purpose

Gets a list of all boxes contained in a graph.

Syntax

```c
#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 deallocated.

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 713.

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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 164.

```c
#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

When compiling a program that uses nvotGetBoxesInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See nvotFreeBox in "nvotFree" on page 226.
- See nvotFreeBoxList in "nvotFree" on page 226.
- See "nvotGetError" on page 234.
- See "nvotGetErrorMsg" on page 237.
nvotGetError

Purpose

Returns the error code set by the last function call.

Syntax

```c
#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_ALREADY_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. Try reissuing the nvotInit routine.

[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 went down during operation. Try reissuing the nvotInit routine.
[NVOT_OVW_CONNECTION_ERROR]
There is an connection error with the NetView program. The connection to the object database cannot be opened.

[NVOT_OVW_TIMED_OUT]
NetView timeout. The timeout value passed to nvotSetSynchronousCreation might not be enough for the complete operation processing, or the connection to the NetView 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:

```c
nvotReturnCode rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
    printf("%s\n", nvotGetErrorMsg (rc));
```

Libraries

When compiling a program that uses nvotGetError, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See [nvotGetErrorMsg](#) on page 237
nvotGetErrorMsg

Purpose

Returns a message string related to a nvotReturnCode.

Syntax

```c
#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 on page 234](#).

```c
nvotReturnCode rc;
If ((rc = nvotGetError()) != NVOT_SUCCESS)
  printf("%s\n", nvotGetErrorMsg (rc));
```

Libraries

When compiling a program that uses nvotGetErrorMsg, link to the /usr/OV/lib/libnvot.lib library.

Files

- `nvot.h`

Related Information

- See [nvotGetError on page 234](#).
nvotGetGraphsInGraph

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 deallocated.

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 *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 was lost during operation. Reissue the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as in the following example:

```c
nvotReturnCode rc;

if ((myGraphs = nvotGetGraphsInGraph (myGraphProt, myGraphName) != NULL) {
    printf ("%s\n", nvotGetErrorMsg (rc));
```

**Examples**

The following example checks all graphs contained in the graph created in the example in [*nvotCreateGraphInGraph* on page 169].

```c
#include <nvot.h>

nvotReturnCode rc;

myGraphList * 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. */
nvodFreeGraphList (myGraphs);
}
/* 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
When compiling a program that uses nvotGetGraphsInGraph, link to the
/usr/OV/lib/libnvot.lib library.

Files
nvot.h

Related Information
- See nvotFreeGraph in nvotFree on page 226.
- See nvotFreeGraphList in nvotFree on page 226.
- See nvotGetError on page 234.
- See nvotGetErrorMsg on page 237.
nvotGetSapsOnVertex

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 NVOT_VERTEX_INVALID_INDEX error code 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 deallocated.

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 713.

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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 189.

```c
#include <nvot.h>

nvotReturnCode rc;

int using = 0;
int providing = 0;

nvotSapList * mySaps = NULL;

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_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()) == 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

   When compiling a program that uses nvotGetSapsOnVertex, link to the
   /usr/OV/lib/libnvot.lib library.

 Files

   nvot.h

 Related Information

   • See nvotFreeSap in [nvotFree" on page 226].
   • See nvotFreeSapList in [nvotFree" on page 226].
   • See [nvotGetError" on page 234].
   • See [nvotGetErrorMsg" on page 237].
nvotGetVerticesInBox

Purpose

Gets a list of all vertices contained in a box graph.

Syntax

```c
#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 NVOT_BOX_INVALID_INDEX error code 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 deallocated.

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 713.

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 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 was lost during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 185:

```c
#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 = nvotGetVerticesInGraph (myBoxProt,
    myBoxName, myVertices->count) != 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));
    }
}
nvotGetVerticesInGraph

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 deallocated.

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, please see "Chapter 3: Using NetView GTM Data Structures" on page 713.

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 was lost during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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" on page 189.

```c
#include <nvot.h>

nvotReturnCode rc;
rvotList * 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. */
vnotFreeVertexList (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

When compiling a program that uses nvotGetVerticesInGraph, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See nvotFreeVertex in [nvotFree](#) on page 226.
- See nvotFreeVertexList in [nvotFree](#) on page 226.
- See [nvotGetError](#) on page 234.
- See [nvotGetErrorMsg](#) on page 237.
nvotInit

Purpose

Initializes the NetView Open Topology interface.

Syntax

```c
#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 gtdm, either locally or remotely. The application using the NetView Open Topology platform can run in one machine while gtdm is be running on another machine.

Hostname is the name of the network node where the gtdm is running. Hostname serves remote connections only. When an application and gtdm are to be running in the same machine, the `hostname` parameter should be set to NULL. Communication between the application and gtdm 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 gtdm’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 gtdm, 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 gtdm 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 gtdm 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 gtdm can associate the vertex it creates with the graph. If gtdm 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 gtdm 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 gtdm 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. Try reissuing the nvotInit routine.

[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. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the routine nvotGetErrorMsg, as shown in the following example:
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.

```c
#include <nvot.h>

nvotReturnCode rc;

gtmHost = NULL;
arcDirection = FALSE;
graphQueryOn = TRUE;

if ((rc = nvotInit (gtmHost, arcDirection, graphQueryOn)) == NVOT_SUCCESS)
    printf("OK : \n", nvotGetErrorMsg (nvotGetError()));
else
    printf("WHOOPS! : \n", nvotGetErrorMsg (nvotGetError()));
```

Libraries

When compiling a program that uses nvotInit, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotDone" on page 224
- See "nvotGetError" on page 234
- See "nvotGetErrorMsg" on page 237
nvotSetCenterBoxForGraph

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 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 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 went down during operation. Try reissuing the nvotInit routine.

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 is displayed 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

When compiling a program that uses nvotSetCenterBoxForGraph, link to the 
/usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotCreateRootGraph" on page 178
- See "nvotCreateBoxInGraph" on page 164
- See "nvotChangeBoxPositionInGraph" on page 110
nvotSetCenterGraphForGraph

Purpose

Specifies what graph symbol is to be the center of a star graph map.

Syntax

```c
#include <nvot.h>

nvotReturnCode nvotSetCenterGraphForGraph (nivotGraphProtocolType graphProtocolParent, char * graphNameParent, nivotGraphProtocolType 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 /usr/OV/conf/oid_to_protocol file.

- **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 went down during operation. Try reissuing the nvotInit routine.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
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 is displayed 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.
", myGraphName, myRootSDLCGraphName);
else
    printf ("An error occurred changing %s symbol position.
", myGraphName);
    printf ("Operation result : %s" , nvotGetErrorMsg (rc));


## Libraries

When compiling a program that uses `nvotSetCenterGraphForGraph`, link to the `/usr/OV/lib/libnvot.lib` library.

## Files

- `nvot.h`

## Related Information

- See ["nvotCreateRootGraph" on page 178](#).
- See ["nvotCreateGraphInGraph" on page 169](#).
- See ["nvotChangeGraphPositionInGraph" on page 125](#).
nvotSetSynchronousCreation

Purpose

Initializes the interface to return OVwObjectIds.

Syntax

```c
#include <nvot.h>
nvotReturnCode nvotSetSynchronousCreation (unsigned synchronous)
```

Description

Objects stored in the gtmd database are also stored in the NetView 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 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_OVW_CONNECTION_ERROR]

There is a connection error with the NetView program. The connection to the NetView object database cannot be opened.

A printable message string is accessible through a call to the nvotGetErrorMsg routine, as shown in the following example:

```c
nvotReturnCode rc;

If ((rc = nvotGetError()) != NVOT_SUCCESS)
    printf ("%s\n", nvotGetErrorMsg (rc));
```

Examples

The following example creates a connection to the NetView object database. After this call, every create function should return the OVwObjectld 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

When compiling a program that uses nvotSetSynchronousCreation, link to the /usr/OV/lib/libnvot.lib library.

Files

nvot.h

Related Information

- See "nvotInit" on page 250
nvplay

**Purpose**

Provide audio when a specific trap is received.

**Syntax**

```
nvplay [-m string] [-r numreps] [ filename.wav ]
```

```
nvplay -h
```

**Description**

The nvplay application can be used to provide audio when a specific trap is received. Use the Trap Settings dialog to configure a wave file to be played when a specific trap is received. This provides quick audio feedback when an important event occurs.

**Flags**

- `-m string`
  
  Displays the specified string in a popup window.

- `-r numreps`
  
  Plays the wave file `numreps` times in a continuous loop.

- `-h`
  
  Displays the usage statement.

The `filename.wav` parameter is the wave file to be played, if it exists.

If multimedia is enabled on your machine, and a microphone is attached, you can record your own wave files. Use the Sound Recorder application:

Start→Programs→Accessories→Multimedia→Sound Recorder

**Examples**

Select **Options→Trap Setting** and type the following line in the Command box in the Trap Properties dialog (assuming `\mydir\nodedown.wav` exists):

```
/usr/ov/bin/nvplay -m "Node %A down" -r 10 \mydir\nodedown.wav
```

When a node down trap is received, a node down message will pop up and the `nodedown.wav` file will be played ten times.

Since the nvplay application uses the desktop, you must specify it as a Windows Application.
nvservice

Purpose

Discover and monitor the services of nodes in the network.

Syntax

nvservice [-options]

Description

The nvservice command enables you to manage the NetView Service from the command line. The NetView Service runs the NetView daemons at system startup and across login sessions. The recommended method to manage the NetView Service (check the status, start, and stop the NetView Service) is to use the Server Setup application (nvsetup).

Flags

The following options are available:

- **install**
  Installs the service.

- **remove**
  Removes the service.

- **debug parameters**
  Runs the NetView Service as a console application for debugging. Use -debug with no parameters to display debugging messages to the \usr\ov\log\ovsmd.log file.

- **abort**
  Exits the NetView Service without first stopping daemons.

- **stop**
  Stops the daemons and exits the NetView Service.

- **help**
  Lists all of the valid command line flags.

- **status**
  Displays the status of the NetView Service.

- **installed**
  Determines whether the NetView Service is installed.
nvsniffer

Purpose
Discover and monitor the services of nodes in the network.

Syntax

Description
The nvsniffer command is used for discovering services on nodes in the network and for monitoring the status of these services. Service objects are created in the object database for each service that is discovered at a node. A node’s services are represented on the map at the node’s interface level. Service objects contain status, and this status optionally contributes to a node’s overall IP status. The node automatically becomes a member of the corresponding service SmartSet for each service that is discovered.

Flags
-c configFile
Optionally specify a configuration file other than the default file located at \usr\ov\conf\nvsniffer.conf. The default configuration file contains several example entries with detailed explanations of how the entries are used by nvsniffer and what the outcome of each entry is after both a Discovery mode run and a Status mode run.

A configuration file may consist of any number of entries, each containing a vertical bar–separated (|) list of fields, one entry per line. Comment lines start with either the // (two slash characters) or a # character in the first column of a line; no comment delimiters are allowed beyond the first column, and no trailing comments in an entry are allowed.

Each field in an entry MUST end with a (|) character, except the last field which must have a terminating new line (carriage return) character. An empty field should be specified as ||. Syntax and format errors are logged in the \usr\ov\log\nv.log file, and the entry in which the error(s) occurred is ignored.

The format of an entry in a configuration file is:
{Node Field}|{TCP Port,,,}|{Service SmartSet}|{Service Label Name}|{Discover Test}|{Status Test}|{Discover Node Selection Criteria}

Where:

{Node Field} is the name of the boolean field to create (if needed) in a node object based on the return status of a Discovery test. This field cannot be empty.

{TCP Port,,,} is a comma-separated list of the TCP port numbers to test on each selected node. Alternatively, this list can consist of any set of values (arguments) if custom tests are specified and they require arguments. This field can be empty ONLY if a custom test is specified in both the Discover Test and Status Test fields.
{Service SmartSet} is the name of the service SmartSet to create (if it does not already exist) during an nvsniffer Discover mode run. This field should be empty if no service SmartSet is to be created.

For nvsniffer Status mode, this field is used as the node selection criteria. If this field is empty, then this entry of the file is ignored during an nvsniffer Status mode.

{Service Label Name} is the label name assigned to the service object created as a result of nvsniffer successfully discovering the service at the nodes being tested. The label name is also stored in the service SmartSet specified in the {Service SmartSet} field. This field can only be blank if the {Service SmartSet} field is blank, and cannot be blank if a {Service SmartSet} field is specified.

{Discover Test} is the optional custom plug-in module to use to discover the service. This must be left empty if there is no custom discovery test. The name of a launchable application or an extension DLL with the NvServiceTest() API is what is specified here. For nvsniffer Status mode, this field is not used. Refer to \usr\ov\doc\nvsniffer.readme for details on how to create and use customs plug-in modules.

{Status Test} is the optional custom plug-in module to be used to status check the already discovered service. This must be left empty if there is no custom status test. The name of a launchable application or an extension DLL with the NvServiceTest() API is what is specified here. The same launchable application or extension DLL with the NvServiceTest( )API can be specified as that specified in the {Discover Test} field. For nvsniffer Discover mode, this field is not used. Refer to \usr\ov\doc\nvsniffer.readme for details on how to create and use customs plug-in modules.

{Discover Node Selection Criteria} specifies which nodes to test during a Discover mode run. It can be either * for all IP nodes, or the name of ANY existing SmartSet. This field is not used during an nvsniffer Status mode run.

All entries in the configuration file that specify * are grouped together so that all discovery tests can be done to an IP node all at once. Entries that specify a SmartSet for this field are processed individually; the node worklist is re-loaded with the members of the specified SmartSet for each test.

A trailing | is not required for this field since this is the last field of an entry. However, each entry (including the last entry in the file) must be terminated with a new line (carriage return) character.

-d Optionally, log the node/service about to be tested to stdout. Useful for diagnosing nodes that are slow to respond. Works best when used with -t 1 flag.

-l logFile
Optionally, specify a log file to store the progress of nvsniffer. All errors are logged to the file \usr\ov\log\nv.log.

-n nodeName
Optionally specify a single node’s Selection Name to test (specify it exactly as it appears in the object database).

This switch can be used for either an nvsniffer Discover mode or Status mode run (-s switch). When specified for a Discover mode run, the service entries in the specified configuration file that have a Discover Node Selection Criteria of *
(for all nodes) are tested, and any entry specifying a SmartSet for the Discover
Node Selection Criteria in which the specified node belongs will be tested. For a
Status mode run, the services that are status checked are those in which the
node is currently a member of the corresponding Service SmartSet.

-o If specified, nodes that have an IP Status of Critical will be tested.

-r minutes
Use this option to reschedule nvsniffer with the current settings and options
every minutes minutes. Valid range is 5..44640 minutes (up to 31 days
maximum).

-s Use this option to have nvsniffer run in Status Mode. Each known service on
the nodes that are members of Service SmartSets are status-checked, as
specified in the configuration file being used. If specified, the custom Status
Test or the NvServiceTest() API in the custom extension DLL is invoked.
Otherwise, a raw port test against the enumerated TCP port numbers specified
in the TCP Port field is performed. Node selection criteria for nvsniffer Status
Mode consists of the nodes that are members of the Service SmartSet specified
in each configuration file entry. Service SmartSets are created by nvsniffer
during a Discover Mode run.

-t numThreads
Optionally specify the maximum number of threads to be activated when
nvsniffer is invoked. The valid range is 1..50, inclusive.

-T timeout
Optionally, specify a timeout for launched custom tests (in seconds). Launched
custom test are normally allowed 120 seconds to complete. Range for timeout
is 1–300 seconds.

-h or -?
Causes the nvsniffer usage information to be displayed.

Examples

nvsniffer -l\usr\ov\log\nvsniffer.log -r 1440 -o

This nvsniffer invocation will discover all services in the default configuration file
located at \usr\ov\conf\nvsniffer.conf and will reschedule itself to discover services
again in 1440 minutes (one day). The progress of nvsniffer will be logged in the file
\usr\ov\log\nvsniffer.log. Any node that has an IP Status of Critical will still be tested
(because of the -o switch).

nvsniffer -r 20 -s

This nvsniffer invocation will status check all services in the default configuration file
located at \usr\ov\conf\nvsniffer.conf and will reschedule itself to status check
services again in 20 minutes.

nvsniffer -n dns.corp.company.com -s -c \usr\ov\conf\nvsniffer2.conf

This nvsniffer invocation will status check all services in the custom configuration
file located at \usr\ov\conf\nvsniffer2.conf against node dns.corp.company.com. Only
the services corresponding to the Service SmartSets that this node belongs to
specified in the configuration file will be status checked.

Files

\usr\ov\conf\nvsniffer.conf
nvSnmpBlockingGetTable

Purpose
Retrieves an entire table from the MIB or an element from the returned table.

Related Functions
nvSnmpGetTableElement
nvSnmpGetTable
nvSnmpWGetTable

Syntax

```c
#include <OV\OVsnmp.h>
#include <OVsnmpApi.h> /*not required for nvSnmpGetTableElement*/

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 nvSnmpWGetTable(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 nvSnmpWGetTable 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 one 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 object ID 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 object ID associated with ifEntry, for example, 1.3.6.1.2.1.2.2.1.

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.

The nvSnmpWGetTable has the same parameters as nvSnmpBlockingGetTable and nvSnmpGetTable. However, if you use nvSnmpWGetTable, you must use the OVsnmpWOpen to manage the transmissions.

The nvSnmpGetTableElement call does not provide any check on the row/column pair passed as parameters. You must check the accuracy of these parameters.
Parameters

session
   Specifies a pointer to a session that was previously established by the application.

oid
   Specifies a pointer to the table object ID.

oidLen
   Specifies the length of the table object ID.

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 will return the number of columns in the 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 column from which a value should be retrieved from the table.

Return Values

If successful, nvSnmpBlockingGetTable returns a pointer to an OVsnmpPdu structure. If unsuccessful, it returns NULL.

If successful, nvSnmpGetTable and nvSnmpWGetTable return a request ID. If unsuccessful, they return −1 (negative one).

If successful, nvSnmpGetTableElement returns an OVsnmpVarBind pointer. If unsuccessful and the PDU sent is not corrupted, it returns NULL.

Error Codes

The nvSnmpBlockingGetTable call returns the error code value in the external variable 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 was 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](#) on page 349 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, link to the following libraries:

- `\usr\OV\lib\libov.lib`
- `\usr\OV\lib\nvsnmp.lib`
- `\usr\OV\lib\libovc.lib`
- `\usr\OV\lib\libovcmapi.lib`

### Related Information

- See [OVsnmpAddVarBind](#) on page 297.
- See [OVsnmpCreatePdu](#) on page 325.
- See [OVsnmpOpen](#) on page 342.
- See [OVsnmpSend](#) on page 349.
nvSnmpErrString

Purpose

Returns SNMP specific error strings.

Syntax

`#include <OV\OVsnmp.h>`

`char *nvSnmpErrString (int error);`

Description

The nvSnmpErrString function returns a textual string that provides information about the error specified in the `error` parameter.

Return Values

The nvSnmpErrString function returns a pointer to a static character string. This string is read only.

If the error number is out of range, the nvSnmpErrString function returns the string “Unknown Error”.

Libraries

When compiling a program that uses the nvSnmpErrString function, link to the following libraries:

- `\usr\OV\lib\libov.lib`
- `\usr\OV\lib\nvsnmp.lib`
- `\usr\OV\lib\libovc.lib`
- `\usr\OV\lib\libovcmapi.lib`
nvsnmptrap

Purpose

Issues an SNMP trap.

Syntax

nvsnmptrap [-d] [-t timeout] [-r retries] [-p port] [-c community] node enterprise
genagent-addr generic-trap specific-trap time-stamp [variable type value ...]

Description

The nvsnmptrap command issues an SNMP trap to node. The node option can be either an internet address or a host name. If you specify a host name, enter the object name using the dot notation. For example, node.org.company.com.

The enterprise variable has the format of A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation. An enterprise value of "" (empty string) is interpreted as the default object ID based on the hardware in use.

An agent-addr variable must be either an Internet address or a host name (see hosts.) An agent-addr option of "" (empty string) is interpreted as the local hostname.

The generic-trap value must be a whole number between zero and six (inclusive).

The specific-trap value can be any 32-bit integer.

The time-stamp value must be a whole number equal to or greater than zero.

Each variable has the format of A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation.

The type must be one of the following types:
  Integer
  Octetstring
  Octetstringhex
  Octetstringoctal
  Octetstringasci
  Objectidentifier
  Null
  Ipaddress
  Counter
  Gauge
  Timeticks
  Opaque
  Opaquehex
  Opaqueoctal
  Opaqueasci

See RFC 1155 for a complete description of each type.

The value must be valid for the specified type. When using a type where a hexadecimal or octal value is needed, the value must have each byte fully defined. For example, fff (or 17377) is not allowed, whereas 0fff (or 017377) is. For type
Null, a value must be specified on the command line, but it is ignored when the request is created. The value must occupy not more than 256 bytes maximum.

If the target node has an error processing the request packet, an error packet returns and a message is displayed to identify how the request was incorrectly formatted.

**Flags**

- `-?` Lists the valid variable types and the usage message.
- `-d` Dumps (lists) input and output packets in a hexadecimal format.
- `-t` Overrides the default timeout with `timeout`, given in 1/10th of a second units.
- `-r` Overrides the default number of retries to `retries`. An exponential backoff algorithm is used, so if timeout was 10 (1 second) and retries was 3, the first timeout would occur at 1 second, the second (first retry) at 2 seconds, the third (second retry) at 4 seconds, and the last (third retry) at 8 seconds. It would take 15 seconds before the command gave up.
- `-p` Overrides the default port for sending and receiving to `port`.
- `-c` Overrides the SNMP community name (as configured in `\usr\ov\conf\ovsnmp.conf`) to `community`. See `ovsnmp.conf` on page 304.

If all arguments are formatted correctly, a message is displayed indicating that the SNMP trap was sent to the remote node.

**Examples**

The following example sends an SNMP trap to node `testnode` using the agent `agentnode` with enterprise equal to:

```
.iso.org.dod.internet.private.enterprises.ibm.ibmProd.netView6000.1
```

And, with time-stamp equal to zero. The trap has generic-trap equal to 6, and specific-trap equal to 1. The information passed with the trap is system.sysDescr.0:

```
nsnmptrap testnode 0 agentnode 6 1 0 system.sysDescr.0 octetstring "AIX 3.2 machine"
```

**Files**

`\usr\ov\conf\ovsnmp.conf`

**Related Information**

- See `snmpwalk` on page 693.
- See `trapd` on page 701.
- See `ovsnmp.conf` on page 304.
- See `snmpget` on page 683.
- See `snmpset` on page 690.
- See `snmpnext` on page 685.

**Request for comments**

`\usr\ov\doc\` (for related RFC’s)
oid_to_protocol

Purpose

Provides a table used to convert from an oid provided in the NetView topology MIB to a string that represents the graph protocol name.

Syntax

oid=protocol name

Description

The oid_to_protocol file is used by the xxmap application to convert from an oid provided in the NetView topology MIB to a string that represents the graph protocol name. The file shipped with the NetView program lists several protocols in an enumeration list. The operator or other applications can add to this file for user-defined oids and protocols. Contact the NetView Association to ensure a unique oid. The oid_to_protocol file has the following format:

```
# Comment
"SNMP ifType"=1.3.6.1.2.1.2.2.1.3

${SNMP ifType}.1="Other"
${SNMP ifType}.2="Regular 1822"
${SNMP ifType}.3="HDH 1822"
${SNMP ifType}.4="DDN X.25"
${SNMP ifType}.5="RFC 877 X.25"
${SNMP ifType}.5="Ethernet CMSACD"
```


oid_to_sym

Purpose

Maps SNMP sysObjectld to graphical user interface symbol types.

Syntax

\usr\ov\conf\c\oid_to_sym

Description

The oid_to_sym file is used by the ipmap command to map the SNMP sysObjectld of a node into the default graphical user interface symbol type used to represent the node in the IP topology submaps it displays. The graphical interface symbol type is composed of a symbol class and symbol sub-class that together make up the symbol type.

Lines in the file contain entries that describe the mapping according to SNMP sysObjectld. Comments are denoted by the # symbol and cause the remainder of the line to be ignored. Blank lines and lines continued to the next line by a backslash (\) at the end of the line are permitted.

Entries in the file contain the following colon-separated fields:

SNMP sysObjectld
The SNMP object ID reported by an agent as the value of the SNMP MIB I system.sysObjectld variable and represented as a series of integers separated by dots (.)

Symbol class
The class to which the symbol belongs. Examples include Connector, Computer, and Device. The default value is Computer.

Symbol subclass
The subclass to which the symbol belongs. Examples for the Computer class include Workstation, Mini, and PC. The default value is Generic.

The values for symbol class and sub-class must match one of the registered symbol types with the graphical interface. For more information on symbol registration, see "OVwRegIntro" on page 591.

Examples

The following entry describes the agent which is provided by IBM for the RS/6000 computer.
1.3.6.1.4.1.11.2.3.2.2:Computer:Workstation # RS/6000

The following entry describes an agent from the XYZ Software Company that runs on a RAD Brouter. The device acts as a bridge and an IP gateway.
1.3.6.1.4.1.999.4.8.1.2:Connector:Brouter # RAD Brouter-XYZ Agent

In the previous example, Brouter would have to be a defined subclass in the Connector class by adding a new symbol definition to the graphical interface symbol registration files for the Brouter subclass.
Files

\usr\ov\conf\oid_to_sym
Language specific symbol mappings

Related Information

- See "ipmap" on page 37
- See "oid_to_type" on page 273
- See "netview" on page 65
- See "OVwRegIntro" on page 591
oid_to_type

Purpose
Maps SNMP sysObjectld to vendor and type.

Syntax
\usr\ov\conf\oid_to_type

Description
The oid_to_type file is used by the netmon daemon to map the SNMP sysObjectld of a node into the correct IP topology properties, as well as the correct vendor and SNMPAgent values for use with the NetView program and the OVW object database. Applications integrated with the NetView program can use the vendor and SNMPAgent values to determine operations that are supported by the agent.

The oid_to_type file contains entries that the user can modify to fit various environments.

Lines in the file contain entries that describe the mapping per SNMP sysObjectld. Comments are denoted by the # symbol and cause the remainder of the line to be ignored. Blank lines are valid, as are lines continued to the next line by ending the line with a backslash (\).

Entries in the file contain the following colon-separated fields:

SNMP sysObjectld
The SNMP object ID reported by an agent as the value of the SNMP MIB I system.sysObjectld variable and represented as a series of integers separated by dots (.)

Note: The fields of the sysObjectld cannot contain the value 0 (zero). This is true whether or not your symbol represents an SNMP_ resource.

vendor
The vendor who makes the device or node (hardware) on which the agent will be executed. The vendor must match one of the enumerated values for the vendor field or remain empty to indicate an unknown vendor.

agent
The type of agent associated with the sysObjectld. The agent must match one of the enumerated values for the SNMPAgent field or remain empty to indicate an unknown agent type.

IP topology properties
The IP topology properties associated with the object. These properties control how the device or node will be treated by the IP-specific components of the system. This field includes a series of characters that represent the types of properties to associate with the given object ID. The following characters are valid, in any order:

G Indicates that the device is a gateway.
B Indicates that the device is a bridge.
H Indicates that the device is a hub or multiport repeater.
I Treats the device as if it did not support SNMP. This would be used to disable SNMP operations to devices whose agents misbehaved in such a way as to cause problems.
The device is a terminal-server-like device with respect to the way its SNMP agent reports the contents of the ip.ipAddrTable for the various IP addresses assigned to it. Some devices report the same IP address in that table regardless of which of the multiple IP addresses assigned to the device are used to make the query.

Creates the device in the "unmanaged" status state.

Assume the device has a default web management page at the URL: http://hostname. The following fields will be created, if necessary, and set:

- isHTTPSupported: TRUE
- isHTTPManaged: TRUE
- ManagementURL: http://hostname

Poll these devices for status using SNMP instead of ICMP pings. Checks both ifAdminStatus and ifOperStatus.

The vendor and SNMPAgent fields are defined through the NetView field registration files during installation. See "OVwRegIntro" on page 591 for more information about on field registration.

Examples

The following entry describes the agent which is provided by IBM for the RS/6000 system computer.

1.3.6.1.4.1.2.3.1.2.1.1.2:IBM:IBM RS/6000: # SNMP agent for AIX 3.2

The following entry describes an agent from the XYZ Software Company that runs on a RAD Brouter. The device acts as a bridge and an IP gateway.

1.3.6.1.4.1.999.4.8.1.2:RAD Brouter:XYZ Software Company:BG

In the previous example, RAD Brouter would have to be a defined vendor and XYZ Software Company would have to be a defined SNMPAgent, as defined by the ovwdb field registration files.

The following entry describes another agent for the XYZ Software Company that runs on several different hardware platforms. Because the hardware vendor cannot be uniquely determined, that field and the IP topology properties field should be left empty.

1.3.6.1.4.1.999.4.8.1.3::XYZ Agent 2: # Unknown vendor

Files

- \usr\ov\conf\oid_to_type
  General mapping of SNMP sysObjectld
- \usr\ov\fields\c\ovw_fields
  Field registration file for the vendor field
- \usr\ov\fields\c\snmp_fields
  Field registration file for the SNMPAgent field

Related Information

- See "netmon" on page 59.
- See "oid_to_sym" on page 273.
- See "ovtopmd" on page 363.
- See "ovwdb" on page 464.
- See "OVwRegIntro" on page 591.
ovaddobj

Purpose

Provides the NetView object registration utility.

Syntax

ovaddobj -t targethost LRF-filename [hostname]

Description

The ovaddobj command performs two functions: it is used to register the information necessary to start up object managers (agents) with the NetView process management daemon ovspmd, and it is used to register in the NetView server’s object registration service database information about the object managers and the objects they represent. The NetView program must be installed on all hosts where objects are registered.

The object manager is identified by a combination of host name and object manager name. If you want to change this information, see “ovdelobj” on page 273. To change other information about the object manager, edit the local registration file (LRF) and rerun ovaddobj. Previous information in the object registration service about this object manager will be replaced by the information in the current LRF.

Note: All objects managed by the object manager must be specified in the same LRF. Running ovaddobj against an LRF containing additional objects managed by a previously registered object manager will not result in adding those objects, but rather in replacing those objects previously registered with these new objects.

Parameters

-t targethost

The targethost flag is used by ovaddobj to expand the ~ (tilde) and ! (exclamation) metacharacters in an instance name of an object in the LRF.

In the case of ~ expansion, the targethost is interpreted as the host on which the object to be accessed resides (target host name). This can be different from hostname, which indicates where the object manager, or agent, resides. The targethost is used for SNMP proxy lookups in the object registration service and for expansion of the ~ character, which can be used in the Object Instance field of the LRF to indicate the IP address of the target host. If the indicated host is assigned multiple IP addresses and the ~ character is used in the Object Instance field, ovaddobj will expand the ~ character with each of the IP addresses, adding an entry to the object registration service for each expansion.

In the case of ! expansion, the object is registered with the ! character in the instance name replaced directly with targethost.

LRF-filename

This field specifies an LRF, which must contain information about a single object manager and the objects it manages. LRF syntax is described in “lrf” on page 48. An LRF may contain objects to be accessed by multiple management protocols.
hostname

This field specifies the remote machine, identified by its host name, where the agent is running.

Examples

**ovaddobj mylrf**

This command registers the object manager and all objects described in the LRF mylrf in the NetView object registration service database. The object manager location is registered to be on the current system.

**ovaddobj mylrf spock**

This command registers the object manager and objects described in the LRF mylrf. The object manager location is registered to be on the system named spock.

**ovaddobj -t system_A mylrf system_B**

If you use this command, the object manager resides on system_B and is proxying for objects on system_A.

Related Information

- See [Lrf](#) on page 48
- See [ovdelobj](#) on page 279
- See [ovspmd](#) on page 354
- See [ovstart](#) on page 358
- See [ovstop](#) on page 361
ovdelobj

Purpose

Provides the NetView object deregistration utility.

Syntax

```
ovdelobj LRF-filename [hostname]
```

Description

The ovdelobj command is used to remove registration information from the NetView object registration-service database. It must be used when an agent is removed, renamed, or relocated to another host system. It need not be used when making other changes to the agent or its managed objects, such as adding or deleting managed objects, changing the startup phase, or changing the path name to the agent executable file.

To change the host system on which an agent resides, issue the ovdelobj command, specifying the LRF file that was used to register the agent and the system on which it was initially installed. Then, issue the ovaddobj command, specifying the same LRF file and the name of the system to which the agent is being moved.

To change the name of the agent, issue the ovdelobj command, specifying the LRF file that was used to register the agent and the system on which it was initially installed. Then, edit the LRF file to change the name of the agent. Next, issue the ovdelobj command, specifying the new LRF file and the name of the system on which it runs.

**Note:** The object registration service enforces permissions on changes to the object registration service database. Permissions are managed by the ORSD running on the object registration service master node.

Parameters

- **LRF-filename**
  Specifies a Local Registration File, which must contain information about a single agent and the objects it manages. LRF syntax is described in page 43. An LRF may contain objects to be accessed by multiple management protocols.

- **hostname**
  Specifies the remote machine, identified by its host name, where the agent is running.

Examples

The following command deletes information in the object registration service about the agent named in the mylrf file, which runs on the current system. The current system is the system on which ovdelobj is being executed.

```
ovdelobj mylrf
```

The following command deletes information in the object registration service about the agent named in the mylrf file, which runs on the system named spock:
ovdelobj my1rf spock

Related Information

- See "lrf" on page 48
- See "ovaddobj" on page 277
- See "ovsnmd" on page 354
- See "ovstart" on page 358
- See "ovstop" on page 361
OVFree

Purpose

Frees the memory allocated by the NetView API.

Syntax

```c
#include <OV\ovw.h>

void OVFree (void *memory_to_free);
```

Description

The OVFree routine frees the character string memory (char *) allocated by NetView API routines. Note that OVFree should not be used to free the character string returned by OVsnmpErrString, because that string is statically allocated.

Parameters

- `memory_to_free`
  
  A pointer to the memory allocated by the NetView API.

Return Values

None

Libraries

When compiling a program that uses OVFree, link to the \usr\ov\lib\libov.lib library.
OVFreeSetupSelect

Purpose

Frees the memory allocated by OVSetupSelect.

Syntax

```c
#include <OV\ovsnmpapi.h>

void OVFreeSetupSelect (struct fd_set *fdset);
```

Description

The OVFreeSetupSelect routine frees the memory allocated by the function
OVSetupSelect for the fd_set data structure.

Parameters

`fdset`

A pointer to the constructed list of open sockets returned by OVSetupSelect.

Return Values

None

Libraries

When compiling a program that uses OVFreeSetupSelect, link to the
\usr\ov\lib\vsnmp.lib library.

Related Information

- See "OVSelect" on page 293.
- See "OVSetupSelect" on page 296.
ovmapcount

Purpose

Checks graphical user interface map reference counts.

Syntax

ovmapcount [-vu]

Description

The ovmapcount command is a troubleshooting command that checks for consistency between the NetView map database and the NetView object database. It checks the values of map reference counts stored in the object database and corrects these values if necessary.

Map reference counts are stored as fields in the object database. The field OVW Maps Exists records the number of maps on which an object exists. The field OVW Maps Managed records the number of maps on which an object is managed. These counts are maintained so that the graphical user interface can determine when an object is deleted from or unmanaged on the last map.

The ovmapcount command reads all of the maps, determines the correct values for the map reference counts, then compares these values with the values stored in the object database. Error messages are printed if an object that exists on at least one map does not exist in the object database or if the value of any of the reference counts stored in the object database are not correct.

Note: The ovmapcount command can take a long time to execute. Because reference count values can change while ovmapcount is executing, the only way to ensure that the analysis is correct is to issue the ovmapcount command when no instances of the ovw command are executing.

Flags

- u Updates incorrect reference-count values that are stored in the object database.
- v Enables verbose output which prints the correct reference counts for each object and the values stored in the object database.

Examples

The following command prints error messages if there are any database inconsistency problems:

```
    ovmapcount
```

The following command updates any map reference counts stored in the object database that are incorrect:

```
    ovmapcount -u
```

Files

```
/usr/ov/databases/openview/mapdb/*
```

  Map database files and directories
Related Information

- See "netview" on page 65.
- See "ovwdb" on page 464.
ovmapdump

Purpose

Prints the contents of the NetView graphical user interface map database.

Syntax

ovmapdump [-v] [-m map] [-s snap] [-lv | -L]

Description

The ovmapdump command prints the contents of the NetView graphical user interface map databases as maintained by the graphical user interface. By default, the ovmapdump command prints a line of textual information for the global map information and a line of textual information for each submap, symbol, and map object represented in the map.

Flags

-  Lists available maps instead of printing a specific map. The list includes the name of each map, the permissions the user has to access that map, the creation time of the map, and the comments associated with the map.
-  Lists available maps in long form. This flag is equivalent to -lv.
-  Uses map as the target map name. The default is default.
-  Prints information about snapshot snap instead of the current information.
-  Enables verbose listing, which involves printing all information associated with the items listed. Information is listed with one line per field associated with an item, with a blank line between each item. Only useful with the -l option.

Examples

You can print the current snapshot of the default map in short form by entering the following command:

ovmapdump

You can print out the snapshot named Snapshot1 of the map MyMap in long form by entering the following command:

ovmapdump -v -m MyMap -s Snapshot1

You can print a list of available maps and their access permissions by entering the following command:

ovmapdump -l

Files

\usr\ov\databases\openview\mapdb*  
  Map database files and directories

\usr\ov\databases\openview\ovwdb*  
  Object database files and directories
Related Information

- See "netview" on page 65.
- See "ovmapsnap" on page 287.
ovmapsnap

Purpose

Creates a snapshot of a map

Syntax

ovmapsnap [-lv] [-m map] -c [-n name] [-C comments] [-m map] -d [-f] [-n name] [-m map]

Description

The ovmapsnap command is used to list, create, and delete snapshots of maps. Snapshots are copies of an existing map at a specific point in time, and can be viewed by using the graphical user interface.

Flags

- **-c** Creates a snapshot of the target map.
- **-d** Deletes the indicated snapshot. If no snapshot is specified with a -n option, the oldest snapshot will be deleted.
- **-f** Forces a deletion with no user interaction. Otherwise, if the standard input is a terminal, the user is prompted before deletion. This flag should be used only with the -d option.
- **-l** Lists the snapshots associated with the target map (default).
- **-m map**
  Uses map as the target map name.
- **-n name**
  Specifies name as the name of the snapshot. Names must be unique across snapshots, and one will be created automatically if necessary when the -c flag is used.
- **-v** Enables verbose listing. This flag should be used only with the -l option.

Examples

You can print a list of snapshots of the map MyMap by entering the following command:

```
ovmapsnap -m MyMap
```

The following NetView at command entry will cause a snapshot to be created of the map MyMap every Saturday at 01:00.

```
/every:S"\usr\ov\bin\ovmapsnap -c -m MyMap"
```

The oldest snapshot associated with the default map can be deleted with no confirmation by entering the following command:

```
ovmapsnap -d -f
```

Files

`\usr\ov\databases\openview\mapdb\*`

Map database files and directories
Object database files and directories

Related Information

- See "netview" on page 65.
- See "ovmapdump" on page 285.
OVmib_get_objid_name

Purpose
Converting a dotted-decimal MIB variable object ID to its textual name.

Syntax
#include <OV\OVsnmp.h>
const char *OVmib_get_objid_name(ObjectID *oid, u_int oid_length);

Description
The OVmib_get_objid_name function 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 are 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 the OVmib_get_objid_name function, link to the \usr\ov\lib\libmib.lib library.
OVmib_read_objid

Purpose
Converts a MIB variable name to its object ID format.

Syntax

```c
#include <OV\OVsnmp.h>
int OVmib_read_objid(const char *name, ObjectID *oid, u_int *oid_length);
```

Description
The OVmib_read_objid function 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
OVmib_read_objid returns 0 (zero) if successful. If the specified MIB variable name could not be found in the MIB tree, it returns 1.

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 the OVmib_read_objid function, link to the `\usr\ov\lib\libmib.lib` library.
ovobjprint

Purpose

Displays the contents of the OVW object database.

Syntax


Description

The ovobjprint command is a troubleshooting tool that enables you to directly view the database maintained by ovwdb. It provides a terminal-based utility with a number of options for displaying the data in the OVW object database. If no options are given, ovobjprint displays all the field values in the database grouped by object ID.

Flags

-S Causes ovobjprint to display a summary of the information stored in the object database. This summary includes information on the number of objects in the database, the number of fields defined in the database and the number of field values in the database.

-o object_id
Causes ovobjprint to display to all the field values set for the object identified by object_id.

-f field_id
Causes ovobjprint to display field information. No field values are displayed using this option. The field information displayed includes Field ID, Field Name, Field Type, and Field Flags. If the optional parameter field_id is supplied, only information concerning that field will be displayed. If field_id is not supplied, information on all fields in the database will be displayed.

-s selection_name
Causes ovobjprint to display field values based on the object name. If the optional parameter selection_name is supplied, all the field values for the object identified by the object name are displayed. If selection_name is not supplied, the object name of every object in the database will be displayed.

-e <field_id>
Causes ovobjprint to display the enumerated values defined for enumerated fields (fields created with the ovwEnumField field type). See "OVwDbCreateField" on page 468 for more information on creating fields of type ovwEnumField. If the optional parameter field_id is supplied, only enumerated values associated with that field are displayed. If field_id is not supplied, the enumeration associated with every enumerated field is displayed.

-c Displays a list of clients attached to ovwdb running on the server.

-l Displays a count of the number of nodes, PCs, workstations, minicomputers, and printers in the database.
Examples

You can display the field values set for each field in the OVW object database by entering the following command:

`ovobjprint`

You can display the object name of every object in the database by entering the following command:

`ovobjprint -s`

Related Information

- See "netview" on page 65
- See "ovwdb" on page 464
- See "OVwDbCreateField" on page 468
OVSelect

Purpose

Selects the socket where data is written or read.

Syntax

```
#include <OV.ovsnmpapi.h>

int OVSelect (int nfds, struct fd_set *readfds, struct fd_set *writelfds,
struct fd_set *exceptfds, struct timeval *timeout);
```

Description

The OVSelect routine determines the status of one socket, waiting if necessary.

Parameters

- `nfds`
  This argument is ignored and included only for the sake of compatibility.

- `readfds`
  An optional pointer to a set of sockets to be checked for readability. Set to
  NULL if you are not interested in checking for readability.

- `writelfds`
  An optional pointer to a set of sockets to be checked for writeability. Set to
  NULL if you are not interested in checking for writeability.

- `exceptfds`
  An optional pointer to a set of sockets to be checked for errors. Set to NULL if
  you are not interested in checking for errors.

- `timeout`
  The maximum time for select to wait, or NULL for blocking operation.

Return Values

If successful, OVSelect returns the total number of sockets that are ready and
contained in the fd_set structures. If failed, OVSelect returns a value of -1 (negative
one). If the time limit expires, OVSelect returns a 0 (zero).

Libraries

When compiling a program that uses OVSelect, link to the \usr\ov\lib\nvsnmp.lib
library.

Related Information

- See "OVFreeSetupSelect" on page 283.
- See "OVSetupSelect" on page 296.
OVSetPDUCommunityName

Purpose
Sets the Community Name field of a PDU structure.

Syntax
```c
#include <OV\ovsnmpapi.h>

void OVSetPDUCommunityName (OVsnmpPdu *pdu, u_char *new_community_name);
```

Description
The OVSetPDUCommunityName routine sets the community name field of a PDU structure created using the OVsnmpCreatePDU function. As a result of changing the community name field, the community length field of the PDU is updated to reflect the length of the new community name.

Parameters
- **pdu**
  A pointer to the PDU structure returned by OVsnmpCreatePDU.
- **new_community_name**
  A pointer to the new community name to be set in the PDU structure.

Return Values
None

Libraries
When compiling a program that uses OVSetPDUCommunityName, link to the \usr\ov\lib\nvsnmp.lib library.

Related Information
- See "OVsnmpAddVarBind" on page 297.
- See "OVsnmpCreatePdu" on page 325.
- See "OVsnmpFixPdu" on page 331.
OVSetPDUIPAddress

Purpose
Sets the IP address field of a PDU structure.

Syntax
```
#include <OV\ovsnmpapi.h>

void OVSetPDUIPAddress (OVsnmpPdu *pdu, struct sockaddr_in *ipaddress);
```

Description
The OVSetPDUIPAddress routine sets the IP address field of a PDU structure created using the OVsnmpCreatePDU function.

Parameters
- **pdu**
  A pointer to the PDU structure returned by OVsnmpCreatePDU.
- **ipaddress**
  A pointer to the new IP address to be set in the PDU structure.

Return Values
None

Libraries
When compiling a program that uses OVSetPDUIPAddress, link to the \usr\ov\lib\nvsnmp.lib library.

Related Information
- See [OVsnmpAddVarBind](#) on page 297.
- See [OVsnmpCreatePdu](#) on page 325.
- See [OVsnmpFixPdu](#) on page 331.
OVSetupSelect

Purpose
Builds a list of sockets in which data is written or read.

Syntax
```
#include <OV\ovsnmpapi.h>

struct fd_set* OVSetupSelect (int *numfds, OVsnmpSession *session_list[]);
```

Description
The OVSetupSelect routine builds a list of open sockets to receive or send data. OVSetupSelect returns fd_set* data that can be used in a subsequent call to OVSelect. Use the OVFreeSetupSelect routine to free memory used by the fd_set structure.

Parameters
- `nfsd`
  A pointer to the number of sockets to be added to the list.
- `session_list[]`
  A pointer to the list of open sessions.

Return Values
OVSetupSelect returns the constructed list of open sockets on which to receive or send data.

Examples
```
read_fds = OVSetupSelect(&num_snmp_fds,session_list);
count = OVSelect (num_snmp_fds, read_fds, NULL, NULL, &tv) ;
  .
  .
  .
  .
  .
  .
  .
  .
  if(read_fds)
     OVFreeSetupSelect(read_fds);
```

Libraries
When compiling a program that uses OVSetupSelect, link to the \usr\ov\lib\nvsnmp.lib library.

Related Information
- See "OVFreeSetupSelect" on page 283
- See "OVSelect" on page 293
**OVsnmpAddVarBind**

**Purpose**

Allocates space for and initializes an OVsnmpVarBind data structure for getting and setting variables.

**Related Functions**

OVsnmpAddNullVarBind  
OVsnmpAddTypedVarBind

**Syntax**

```c
#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_len` 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_len` 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 ID 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 `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 \us\ov\include\OVsnmpAsn1.h type that will be assigned to
the variable. Valid types are provided in the \us\ov\include\OVsnmpAsn1.h 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, the OVsnmpAddVarBind functions return a pointer to the new
OVsnmpVarBind structure that was added to the OVsnmpPdu structure. If
unsuccessful, they return NULL.

Error Codes
The OVsnmpAddVarBind functions return the error code value in the external
variable OVsnmpErrno. If the SNMP_SYSERR_MALLOC value is returned, 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 happens when the
value type is INTEGER and val_len is not equal to one.

Libraries
When compiling a program that uses OVsnmpAddVarBind or one of its related
functions, link to the following libraries:
• \us\ov\lib\nvsnmp.lib
• \us\ov\lib\libov.lib
• \use\ov\lib\libntl.lib

Related Information
• See “OVsnmpIntro” on page 337
• See “OVsnmpOpen” on page 342
• See “OVsnmpCreatePdu” on page 325
• See “OVsnmpDoRetry” on page 327
• See “OVsnmpSend” on page 348
• See “OVsnmpFreePdu” on page 333
• See “OVsnmpSend” on page 349
• See “OVsnmpBlockingSend” on page 295
**OVsnmpBlockingSend**

**Purpose**

Sends an SNMP PDU and waits for the response.

**Syntax**

```c
#include <OV\OVsnmp.h>

OVsnmpPdu *OVsnmpBlockingSend(OVsnmpSession *session, OVsnmpPdu *pdu)
```

**Description**

The OVsnmpBlockingSend function 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 timeout occurs.

Timeouts occur for a number of reasons, such as when the destination host is not providing SNMP services, networks are congested, or the destination host is not responding. Time-outs do not indicate that an error in processing has occurred.

The timeout 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 function succeeds, the PDU specified in the `pdu` parameter is freed by a call to the OVsnmpFreePdu function. 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 OVsnmpWOpen function. Use the OVsnmpWSend command instead.

**Parameters**

*session*

Specifies a pointer to a valid OVsnmpSession structure returned by a call to the OVsnmpOpen function. 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 the OVsnmpCreatePdu function. The *pdu* structure contains the PDU type and a pointer to the OVsnmpVarBind list.

**Return Values**

If successful, the OVsnmpBlockingSend function returns a pointer to an OVsnmpPdu structure that contains the response to the outbound PDU. If unsuccessful, it returns NULL.

**Error Codes**

The OVsnmpBlockingSend function returns the error code value in the external variable 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 `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 unnecessary amounts of memory.

**Libraries**

When compiling a program that uses OVsnmpBlockingSend, link to the following libraries:
- `\usr\ov\lib\nvsnp.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

**Related Information**

- See "OVsnmpIntro" on page 337.
- See "OVsnmpOpen" on page 342.
- See "OVsnmpClose" on page 302.
- See "OVsnmpCreatePdu" on page 325.
- See "OVsnmpAddVarBind" on page 297.
- See "OVsnmpSend" on page 349.
- See "OVsnmpFreePdu" on page 333.
- See OVsnmpWOOpen in "OVsnmpOpen" on page 342.
- See OVsnmpWSend in "OVsnmpSend" on page 349.
- See "OVsnmpOpen" on page 342.
OVsnmpClose

Purpose
Ends an SNMP session and frees resources allocated by the session.

Related Functions
OVsnmpWClose

Syntax
```
#include <OV\OVsnmp.h>

int OVsnmpClose(OVsnmpSession *session)
int OVsnmpWClose(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 OVsnmpWClose 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. OVsnmpWClose must be used when session was created by OVsnmpWOpen.

Parameters

- `session`
  Specifies a pointer to an OVsnmpSession structure that was allocated by a call to OVsnmpOpen or OVsnmpWOpen,

Return Values
If successful, the OVsnmpClose and OVsnmpWClose functions return 0 (zero). If unsuccessful, they return −1 (negative one).

Error Codes
The OVsnmpOpenClose and OVsnmpWClose functions return the error code value in the external variable `OVsnmpErrno`.

The following is a possible error:

```
[SNMP_ERR_BAD_SESSION]
   Session was not created by OVsnmpOpen or OVsnmpWOpen.
```

Libraries
When compiling a program that uses OVsnmpClose or OVsnmpWClose, link to the following libraries:
- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information
- See "OVsnmpIntro" on page 337
- See "OVsnmpOpen" on page 342
- See "OVsnmpSend" on page 349.
- See OVsnmpWOpen in "OVsnmpSend" on page 349.
ovsnmp.conf

Purpose

Contains NetView SNMP and status polling configuration.

Description

The `/usr/ov/const/ovsnmp.conf` file associates node names or IP addresses with SNMP community names, timeout intervals, retries, and proxies. Additionally, the `ovsnmp.conf` file contains the NetView node status monitoring interval. The `ovsnmp` and `nvsnmp` libraries and all NetView applications use this file to determine the parameters to use for SNMP requests.

Each line in the `ovsnmp.conf` file has the following format:

```
target:community:proxy:timeoutInterval:retry:pollInterval:set-community:
```

Blank lines are ignored. A `#` character indicates the beginning of a comment; characters from the `#` character to the end of the line are ignored. An empty (defaulted) field will use the first pattern-matching character entry found that specifies the defaulted field and matches the node name. Any negative integer value will be ignored and treated as a default. A match is determined if the requested node’s official (first) IP address matches the range specified by an IP pattern-matching character specification. The absolute default target (`*.*.*.*`), will match any node name, and will fill in remaining default parameters. This default target will also match any open node name. The default target must exist in the `ovsnmp.conf` file, and have no fields defaulted.

The `target` is the only required field. The target field takes the form of either a node name or an IP address pattern (IP address or IP pattern-matching character). IP pattern-matching characters are defined using characters from the set `[0123456789-]`, such as 15.6.80.* or 15.6.80-83.*. This target is matched against the node name of the object to which the SNMP request is directed. The `*` character is a shorthand for the range 0–255. Beware of overlapping pattern-matching character specifications, because the first match in the file for the given node name will be used to fill defaults. If some fields are still defaulted, further matches will fill in defaulted fields. Finally, the default target, `*.*.*.*` will fill in any remaining fields.

The `community` field specifies the community string used for the SNMP request. Community names are similar to passwords, in that they authorize access to MIB variables. The community name sent with a request must match the community name expected by the target SNMP node (agent). Some SNMP nodes support multiple levels of access based on the access level of the community name (read-only or read-write). Because the capability of changing a remote node’s configuration might be sensitive, store only read-only SNMP community names in the `ovsnmp.conf` file.

The `proxy` field specifies the node used to proxy the request. Proxies are used to enable SNMP to access nodes that do not support SNMP. A proxy host usually receives the SNMP request and forwards it to the requested node using a non-SNMP protocol. In a typical proxy environment, the community proxy name is used to determine the non-SNMP node destination for the request. If the `*` character is used as a proxy name, it indicates that a proxy will not be performed. This is especially useful when the target specifies a range.
The **timeoutInterval** field specifies the (non-negative) number of one-tenth of a second before a first retry. An exponential algorithm is used, so 3 retries and a timeout interval of 5 will take 0.5, 1.0, 2.0, and 4.0 for a total of 7.5 seconds until the SNMP command gives up. Note that a timeout interval value of 0 (zero) is allowed, but is rounded to 1, so the exponential algorithm will be effective.

The **retry** field is the (non-negative) number of retries to attempt before returning an error.

The **pollInterval** field specifies the number of seconds between network monitor status polling of a node.

The **set-community** field specifies a community name for use as authorization when issuing SNMP set requests. Sometimes SNMP agents are configured with different community names for set requests and get requests. Use set-community to specify a different name. The default for set-community is public. The MIB Browser uses the set-community name when performing set requests.

A node name must be the official host name. If the target field is a host name, the first IP address of that node will be used for IP address matching.

**Note:** A gethostbyname call is made for each node name in the ovsnmp.conf file every time it is loaded. If there are many entries in the file, commands can take longer to perform their first SNMP request. To avoid this problem, use either IP addresses or IP address ranges. When you use IP addresses, only that address is added, and no gethostbyname call is made. When you use node names, all addresses for that node are added.

The NetView graphical user interface enables you to edit or view this file. This file can be accessed by using the Options..SNMP menu item.

### Examples

The following lines show an example \usr\ov\conf\ovsnmp.conf file:

```plaintext
15.6.80.1–5:admin:*:30:0:60:
15.6.80.6:passwd:*::30:
nodeA:passwd:*:::
15.6.80.10–255:pass:*:25:5:7:

# # open node which does not support SNMP
# open_node:proxypw:15.6.80.11:::
# # default parameters for network 15
# 15.*.*.*:public:*:20:2:300:
# default parameters for all other networks
*.*.*.*:public::50:3:3600:
```

Table 10 shows the effect of the examples in tabular format.

<table>
<thead>
<tr>
<th>Nodename\IP</th>
<th>Community</th>
<th>Timeout Interval</th>
<th>Retry</th>
<th>Poll Interval</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.6.80.1</td>
<td>admin</td>
<td>30</td>
<td>0</td>
<td>60</td>
<td>15.6.80.1</td>
</tr>
<tr>
<td>nodeA</td>
<td>passwd</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>15.6.80.6</td>
</tr>
<tr>
<td>15.6.80.10</td>
<td>pass</td>
<td>25</td>
<td>5</td>
<td>7</td>
<td>15.6.80.10</td>
</tr>
</tbody>
</table>
Table 10. Effect of Sample \usr\ov\conf\ovsnmp.conf file (continued)

<table>
<thead>
<tr>
<th>Nodename\P</th>
<th>Community</th>
<th>Timeout Interval</th>
<th>Retry</th>
<th>Poll Interval</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_node</td>
<td>proxypw</td>
<td>50</td>
<td>3</td>
<td>3600</td>
<td>15.6.80.11</td>
</tr>
</tbody>
</table>

Files

_files\\usr\ov\conf\ovsnmp.conf_

Related Information

- See `snmpwalk` on page 692
- See `nvsnmptrap` on page 270
- See `OVsnmpIntro` on page 337
OVsnmpConfAllocEntry

Purpose

Allocates dynamic storage for an OVsnmpConfEntry structure.

Syntax

```c
#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/nvsnmp.lib`
- `/usr/ov/lib/libov.lib`

Related Information

- [OVsnmpConfFreeEntry](#) on page 312
OVsnmpConfClose

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\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information
- "OVsnmpConfOpen" on page 313
**OVsnmpConfCopyEntry**

**Purpose**

Allocates a new OVsnmpConfEntry and copies the contents of the old OVsnmpConfEntry to the new one.

**Syntax**

```c
#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 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\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

**Related Information**

- "OVsnmpConfFreeEntry" on page 313
- "ovsnmp.conf" on page 304
OVsnmpConfDeleteEntry

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 must be called prior to this call.

Libraries
When compiling a program that uses this routine, link to the following libraries:
- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

Related Information
- [OVsnmpConfOpen](#) on page 313
- [ovsnmp.conf](#) on page 304
OVsnmpConfFreeDest

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 on page 337] 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\nvsnmp.lib
• \usr\ov\lib\libov.lib

Related Information
• [OVsnmpConfResolveDest on page 319]
OVsnmpConfFreeEntry

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,
- OVsnmpConfReadNextEntry,
- OVsnmpConfCopyEntry,
- OVsnmpConfParseEntry,
- OVsnmpConfAllocEntry,
- OVsnmpConfReadDefault
- ovsnmp.conf

Parameters

ce A pointer to the OVsnmpConfEntry structure that is to be freed. See "OVsnmpIntro on page 337" 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/ovsnmp.lib
- /usr/ov/lib/libov.lib

Related Information

- "OVsnmpConfReadEntry" on page 317
- "OVsnmpConfCopyEntry" on page 309
- "OVsnmpConfAllocEntry" on page 307
- "OVsnmpConfReadDefault" on page 316
OVsnmpConfOpen

Purpose

Opens a SNMP Configuration database for subsequent use.

Syntax

```c
#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, 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 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 shadow file if the database is modified. This flag overrides whatever is specified in the database control record concerning compatibility.

Return Values

0 if successful; -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 shadow configuration file from which the database will be created.

[SNMP_ERR_DB_INVALID_TIMEOUT]
The shadow file contains an entry with an invalid timeout value.

[SNMP_ERR_DB_INVALID_RETRY]
The shadow file contains an entry with an invalid retry value.

[SNMP_ERR_DB_INVALID_POLL_INTERVAL]
The shadow file contains an entry with an invalid poll interval value.

[SNMP_ERR_DB_INVALID_REMOTE_PORT]
The shadow file contains an entry with an invalid remote port value.

[SNMP_ERR_DB_INVALID_NAME]
The shadow file contains an entry with an invalid target name.

[SNMP_ERR_DB_COMMUNITY_TOO_LONG]
The shadow file contains an entry with a community string with greater than MAX_COMMUNITY_LEN (255) characters.

[SNMP_ERR_DB_INVALID_WILDCARD]
The shadow file contains an entry with an invalid wildcard specification.

[SNMP_ERR_DB_OVERWRITE_ERROR]
The shadow file contains conflicting entries.

[SNMP_ERR_DB_WRITE_ERROR]
The database cannot be written.

[SNMP_ERR_INVALIDHOST]
The shadow 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 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\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information
- "OVsnmpConfClose" on page 308
- "ovsnmp.conf" on page 304
OVsnmpConfReadDefault

Purpose

Reads the global default parameters in the SNMP Configuration Database.

Syntax

```c
#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.

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 on page 337](#) 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 function.

Dependencies

The database must be opened with OVsnmpConfOpen before this function is used.

Libraries

When compiling a program that uses this routine, link to the following libraries:
- `\usr\ov\lib\nsnmp.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- [OVsnmpConfStoreDefault](#) on page 321
- [OVsnmpConfFreeEntry](#) on page 312
- [ovsnmp.conf](#) on page 304
OVsnmpConfReadEntry

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" on page 304.

The memory to which the return pointer refers is dynamically allocated. It should be freed by the caller using the OVsnmpFreeEntry 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" on page 337 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 before this function is used.

Libraries
When compiling a program that uses this routine, link to the following libraries:
• \usr\ov\lib\nvsnmp.lib
Related Information

- "OVsnmpConfStoreEntry" on page 323
- "OVsnmpConfFreeEntry" on page 312
- "ovsnmp.conf" on page 304
OVsnmpConfResolveDest

Purpose

Returns the resolved SNMP configuration parameters for a target node.

Syntax

```c
#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.

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 before
  this function is used.

Libraries
  When compiling a program that uses this routine, link to the following libraries:
  • `\usr\ov\lib\nvsnmp.lib`
  • `\usr\ov\lib\libov.lib`

Related Information
  • "OVsnmpConfOpen" on page 313
  • "OVsnmpConfFreeDest" on page 311
OVsnmpConfStoreDefault

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 operation after wildcards have been applied.

Parameters
ce A pointer to an OVsnmpConfEntry structure which contains the global default parameters.

The OVsnmpConfEntry is described in "OVsnmpIntro" on page 337.

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 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_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 shadow 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\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information
- "OVsnmpConfStoreEntry" on page 323
- "ovsnmp.conf" on page 304
OVsnmpConfStoreEntry

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 operation after wildcards have been applied.

For more information on the OVsnmpConfEntry data structure, see "OVsnmpIntro" on page 337 or the Tivoli NetView for Windows NT 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 shadow 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 before this function is used.

Libraries
When compiling a program that uses this routine, link to the following libraries:
- \usr\ov\lib\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information
- "OVsnmpConfOpen" on page 313
- "ovsnmp.conf" on page 304
OVsnmpCreatePdu

Purpose
Allocates a OVsnmpPdu data structure of the specified message type.

Syntax
```c
#include <OV\OVsnmpapi.h>
OVsnmpPdu *OVsnmpCreatePdu(int type)
```

Description
The OVsnmpCreatePdu function 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. Free it with a call to the OVsnmpFreePdu function, unless it is freed by the OVsnmpSend or OVsnmpBlockingSend functions 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 function and add variables to the OVsnmpPdu structure with calls to either the OVsnmpAddNullVarBind or the OVsnmpAddTypedVarBind functions.

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 in the external variable OVsnmpErrno. If SNMP_SYSERR_MALLOC is returned, 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, link to the following libraries:
- `\usr\ov\lib\invsnmp.lib`
• \usr\ov\lib\libov.lib

Related Information

• See "OVsnmpAddVarBind" on page 297.
• See "OVsnmpBlockingSend" on page 299.
• See "OVsnmpFixPdu" on page 331.
• See "OVsnmpFreePdu" on page 333.
• See "OVsnmpIntro" on page 337.
• See "OVsnmpOpen" on page 342.
• See "OVsnmpSend" on page 349.
OVsnmpDoRetry

Purpose
Retransmits pending SNMP requests.

Syntax

```c
#include <OV\OVsnmp.h>
#include <sys\time.h>

void OVsnmpDoRetry( )
```

Description

The OVsnmpDoRetry function is intended to be used with the OVsnmpGetRetryInfo function and select when you are using the ovsnmp or the nvsnmp library in a nonblocking manner. The calling process uses OVsnmpDoRetry to retransmit pending SNMP requests.

The OVsnmpDoRetry function 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 supplied on the call to the OVsnmpOpen function, the function specified by the `callback` parameter supplied on the call to the OVsnmpOpen function is called with the `command` parameter set to SNMP_ERR_NO_RESPONSE.

Return Values

The OVsnmpDoRetry function returns NULL.

Error Codes

There are no errors returned by the OVsnmpDoRetry function.

Libraries

When you are compiling a program that uses the OVsnmpDoRetry function, link to the following libraries:
- \usr\ov\lib\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information

- See "OVsnmpIntro" on page 337.
- See "OVsnmpGetRetryInfo" on page 335.
- See "OVsnmpOpen" on page 342.
- See "OVsnmpRead" on page 345.
- See "OVsnmpSend" on page 349.
OVsnmpErrString

Purpose

Returns SNMP specific error strings.

Syntax

```c
#include <OV\OVsnmp.h>

char *OVsnmpErrString(int error)
```

Description

The OVsnmpErrString function returns a textual string that provides information about the SNMP-specific error specified in the `error` parameter. If the error number is out of range, the string `Unknown Error` is returned.

Return Values

The OVsnmpErrString function returns a pointer to a static character string. This string is read-only.

Libraries

When compiling a program that uses the OVsnmpErrString function, link to the following libraries:
- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "OVsnmpIntro" on page 337.
- See "OVsnmpOpen" on page 342.
- See "OVsnmpClose" on page 302.
- See "OVsnmpAddVarBind" on page 297.
- See "OVsnmpBlockingSend" on page 299.
- See "OVsnmpCreatePdu" on page 325.
- See "OVsnmpFixPdu" on page 331.
- See "OVsnmpRecv" on page 347.
- See "OVsnmpSend" on page 349.
- See "OVsnmpTrapOpen" on page 352.
OVsnmpEventOpen

Purpose

Connects to the trapd daemon and sets up to receive filtered traps.

Syntax

```
#include <OV\OVsnmp.h>

OVsnmpSession *OVsnmpEventOpen (const char *peername,
    const char *entityName,
    void (*callback)(), void
    void *callback_data,
    const char *filter);
```

Note that the arguments peername and entityName are ignored; they are included only for compatibility.

Dependencies

The OVsnmpEventOpen function is dependent on the trapd daemon. If the trapd daemon is not running, the call to the OVsnmpEventOpen function will fail.

Description

The OVsnmpEventOpen function creates an active SNMP session that is used explicitly for receiving filtered traps. You cannot send any data and can only receive traps on this session. The new session communicates with the trapd daemon and receives filtered traps that are sent to the trapd daemon. The new session can be used with the OVsnmpRead or OVsnmpRecv functions 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 called to process the inbound trap. It is an error to use OVsnmpRead with a NULL callback parameter.

If the calling process will not use callback functions, the OVsnmpRecv function must be used to receive traps. OVsnmpRecv returns a pointer to the trap PDU that was received. It does not use the callback function.

Parameters

`callback`

A pointer to the routine that will be called to process inbound traps if the OVsnmpRead function is used to receive the trap. In order for the calling process to use callback functions, this parameter must point to a valid function. Specifies a pointer to the community name that will be used in sending requests.

`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.

`filter`

The filter parameter is an expression describing which events pass the filter. It consists of an enterprise OID optionally followed by a specific event specification. If both parts are present in the filter they are separated by a zero.
The following examples of the filter parameter illustrate its use:

```
".1.3.6.1.4.1.2.6.*" // accept all NetView enterprise traps plus any enterprise beginning with .1.3.6.1.4.1.2.6

".1.3.6.1.4.1.2.6.*.0.1" // accept only events with specific event of (1) in the NetView enterprise

".1.3.6.1.4.1.2.6.*.0.1-5" // accept a range of events in the NetView enterprise with specific ids 1 through 5

".1.3.6.1.4.1.2.6.*.0.1-5,7" // construct a disjoint range of events
```

Return Values

If successful, the OVsnmpEventOpen function returns a pointer to a new OVsnmpSession structure. Any memory allocated for the OVsnmpSession structure must be freed by calling the OVsnmpClose function.

If unsuccessful, the OVsnmpEventOpen function returns NULL. Note: The trapd daemon might stop after the OVsnmpEventOpen function 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

The OVsnmpEventOpen function returns the error code value in the external variable OVsnmpErrno. If one of the SNMP_SYSERR_* values is returned, the global variable errno contains the error code returned by the failed system call.

The following list describes the possible errors:

Libraries

When compiling a program that uses OVsnmpEventOpen, link to the following libraries:
- /usr/ov/lib/nvsnmp.lib
- /usr/ov/lib/libov.lib

Related Information

- See "OVsnmpOpen" on page 342
- See "OVsnmpClose" on page 302
- See "trapd" on page 700
- See "OVsnmpRead" on page 345
- See "OVsnmpRecv" on page 347
- See "OVsnmpTrapOpen" on page 352
**OVsnmpFixPdu**

**Purpose**
Deletes a variable with an error from an SNMP PDU.

**Syntax**
```
#include <OV\OVsnmp.h>
OVsnmpPdu *OVsnmpFixPdu(struct OVsnmpPdu *pdu, int type)
```

**Description**
The OVsnmpFixPdu function deletes a variable with an error from the supplied `pdu` and creates a new OVsnmpPdu structure. The OVsnmpPdu data structure that is returned will be the specified type 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 supplied `pdu`.

The OVsnmpFixPdu function should be called when a response is received and the `error_status` variable is not equal to SNMP_ERR_NOERROR.

The OVsnmpFixPdu function 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 when OVsnmpFixPdu is called.

On successful return from OVsnmpFixPdu, the supplied `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 returned PDU. 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 supplied `pdu` is freed, so this memory must be dynamic. For a further discussion of dynamic memory, see [Description on page 333] in [OVsnmpFreePdu on page 333].
Error Codes

Upon failure, the OVsnmpFixPdu function returns the error code value in the external variable OVsnmpErrno. If SNMP_SYSERR_MALLOC is returned, 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 for new PDU is not valid.

**[SNMP_ERR_BAD_PDU]**

The supplied 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 supplied pdu.

**[SNMP_ERR_NO_ERRS]**

There is no error in the supplied pdu.

Libraries

When compiling a program that uses OVsnmpFixPdu, link to the following libraries:

- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See [OVsnmpIntro](#) on page 337.
- See [OVsnmpCreatePdu](#) on page 325.
- See OVsnmpAddNullVarBind in [OVsnmpAddVarBind](#) on page 297.
- See [OVsnmpSend](#) on page 349.
- See [OVsnmpBlockingSend](#) on page 299.
- See [OVsnmpRecv](#) on page 347.
- See [OVsnmpRead](#) on page 345.
OVsnmpFreePdu

Purpose

Frees all memory associated with the specified PDU.

Syntax

```c
#include <OV\OVsnmpapi.h>

void OVsnmpFreePdu(OVsnmpPdu *pdu)
```

Description

The OVsnmpFreePdu function frees all memory associated with the supplied `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 SNMP API functions 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 by those functions. In this case, it is the caller’s responsibility to free the OVsnmpPdu structure with a call to the OVsnmpFreePdu function.

Note: All memory associated with the supplied `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

The OVsnmpFreePdu function does not return a value.

Libraries

When compiling a program that uses the OVsnmpFreePdu function, link to the following libraries:

- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "OVsnmpIntro" on page 337
- See "OVsnmpCreatePdu" on page 325
- See "OVsnmpAddVarBind" on page 297
- See "OVsnmpRecv" on page 347
- See "OVsnmpRead" on page 345
• See [OVsnmpFixPdu](#) on page 331.
OVsnmpGetRetryInfo

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 function is intended to be used with select and the
OVsnmpDoRetry function when you are using the 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 interval in which 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 find 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 function in the context of a nonblocking 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**
  
  If there are pending SNMP requests, as indicated by the return value from
  OVsnmpGetRetryInfo, tvp specifies the structure indicating the time interval in
  which the next retry should be done. Select this value as the timeout parameter.

  **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

The OVsnmpGetRetryInfo function returns the number of SNMP related file
descriptors that have pending requests on them. Select this value as the nfds
parameter 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. If used as the timeout parameter to select, MAX_ALARM will cause the select call to block indefinitely.

Error Codes

There are no errors returned by the OVsnmpGetRetryInfo function.

Libraries

When compiling a program that uses OVsnmpGetRetryInfo, link to the following libraries:
- \usr\ov\lib\nsnmp.lib
- \usr\ov\lib\libov.lib

Related Information

- See “OVsnmpIntro” on page 337
- See “OVsnmpOpen” on page 342
- See “OVsnmpClose” on page 302
- See “OVsnmpDoRetry” on page 327
- See “OVsnmpSend” on page 349
- See “OVsnmpRead” on page 345
OVsnmpIntro

Purpose

Provides an introduction to the ovsnmp library.

Description

The NetView SNMP Application Programming Interface (API) library is provided for applications that perform network management functions using the SNMP options. The API functions send and receive SNMP Protocol Data Units (PDUs) and perform 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 reference pages listed in [Related Information on page 341] 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: blocking and nonblocking.

Use the OVsnmpSend function 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 timeout occurs.

Use the OVsnmpBlockingSend function 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 the OVsnmpRecv function to receive a PDU on a single session. This function 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 the OVsnmpRead function 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 functions to retransmit PDUs.

The interval between retransmissions and the number of retransmissions are indicated by the appropriate parameters to the OVsnmpOpen function.
Location Transparency

The nvsnmp 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 option pull-down menu when running NetView graphical user interface. Information about proxies is stored in the ovsnmp.conf file. See "ovsnmp.conf" on page 304.

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 \usr\ov\include\OVsnmpApi.h.

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. Allocate an OVsnmpSession structure with a call to the OVsnmpOpen function 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 the OVsnmpClose function, the memory pointed to by the community parameter 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 field as returned by strlen.

- **int sock_fd**
  The file descriptor used in sends and receives for this session.

- **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 by the OVsnmpRead function.

  **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_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
the OVsnmpRecvTraps function, but it is permitted for an
application to receive traps on a private port.

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 the
OVsnmpCreatePdu and OVsnmpFreePdu functions, respectively.

The OVsnmpPdu 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 address will contain the address of the host that sent the
PDU.

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 functions. You should not should 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 the
OVsnmpSend, OVsnmpFreePdu, OVsnmpFixPdu, or OVsnmpBlockingSend
functions.

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.
The system uptime for the agent that sent the trap.

Community for incoming PDUs. PDU-specific community names are used as follows:
- On outbound PDUs, if community is non-NULL, it overrides the default community for the session for this PDU transaction only. If used, the string pointed to by community must be dynamically allocated.
- On inbound PDUs, community is always set to the community string in the received message. This is needed so that an application can perform community-specific processing on inbound messages, especially traps.

Length of community name

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 the OVsnmpSend, OVsnmpFreePdu, OVsnmpFixPdu, or OVsnmpBlockingSend functions.

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 ID for this variable. This is dynamic memory.
- u_int oid_length: The number of elements or sub-identifiers in the oid.
- 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 val. This might not be the number of bytes in val.

This is a union that contains pointers to the actual value of the variable. This union includes the following fields:

- long *integer: If type is integer, this variable contains a pointer to the integer value.
- u_char *string: If type is string, this variable contains a pointer to the string value.
- ObjectID *objid: If type is objid, this variable contains a pointer to the object ID value.

Callback functions in nonblocking mode
The callback function that is registered in the OVsnmpSession data structure will be invoked as shown:

\[(\text{void}) \text{callback} \ (\text{type}, \ \text{session}, \ \text{requid}, \ \text{pdu}, \ \text{data})\]

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
- **requid**: The request ID of the PDU
- **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 refer to this data again.

**Libraries**

When compiling a program that uses the NetView SNMP API, link to the following libraries:

- `/usr/ov/lib/nvsnmp.lib`
- `/usr/ov/lib/lovov.lib`

**Implementation Specifics**

The ovsnmp library is based on version 1.1 of the CMU library.

**Files**

- `ovsnmp.conf`
- `OVsnmpApi.h`

**Related Information**

- See [OVsnmpAddVarBind](#) on page 297
- See [OVsnmpBlockingSend](#) on page 299
- See [OVsnmpClose](#) on page 302
- See [OVsnmpCreatePdu](#) on page 325
- See [OVsnmpDoRetry](#) on page 327
- See [OVsnmpErrString](#) on page 328
- See [OVsnmpFixPdu](#) on page 331
- See [OVsnmpFreePdu](#) on page 333
- See [OVsnmpGetRetryInfo](#) on page 335
- See [OVsnmpOpen](#) on page 342
- See [OVsnmpRead](#) on page 345
- See [OVsnmpRecv](#) on page 347
- See [OVsnmpSend](#) on page 349
- See [OVsnmpTrapOpen](#) on page 352
OVsnmpOpen

Purpose

Establishes an active SNMP session for communication with an SNMP agent.

Related Functions

OVsnmpWOpen

Syntax

OVsnmpSession *OVsnmpOpen (char *community, char *peername, int retries, int interval, u_short local_port, u_short remote_port, void (*callback)(), void *callback_data);

OVsnmpSession *OVsnmpWOpen (char *community, char *peername, int retries, int interval, short local_port, short remote_port, void (*callback)(), void *callback_data);

Description

The OVsnmpOpen and OVsnmpWOpen functions 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 the OVsnmpOpen or OVsnmpWOpen functions 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 OVsnmpWOpen functions 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 calling the OVsnmpGetRetryInfo function 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 call the OVsnmpDoRetry function to retransmit the request. If data arrives before the select call times out, the calling process should call the OVsnmpRead or OVsnmpRecv functions to receive the response.

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" on page 304). 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 pointer is specified using either an IP address or a host name. If peername is NULL or of zero length, an error occurs.
This is the desired destination for all requests. This pointer is specified using either an IP address, a host name, "local host", or the exact local system name. If peername is NULL or zero, an error occurs.

`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 sockets call. For the calling process to receive traps on the OVsnmpSession socket rather than using the OVsnmpTrapOpen function, 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 called when the OVsnmpRead function receives a PDU 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 `callback` 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, the OVsnmpOpen function returns a pointer to a new OVsnmpSession structure. Any memory allocated for the OVsnmpSession structure must be freed by calling the OVsnmpClose function.

If unsuccessful, the OVsnmpOpen function returns NULL.

**Error Codes**

The OVsnmpOpen function returns the error code value in the external variable OVsnmpErrno. If one of the SNMP_SYSERR_* values is returned, 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_SYSERRCONNECT]
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.

[SNMP_ERR_GETHOSTBYNAME]
The destination specified by the peername parameter is not an IP address or is not in the host database.

Libraries

When compiling a program that uses the OVsnmpOpen and OVsnmpWOpen functions, link to the following libraries:

- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

Files

ovsnmp.conf

Related Information

- See “OVsnmpClose” on page 303
- See “OVsnmpSend” on page 343
- See “OVsnmpIntro” on page 337
- See “OVsnmpGetRetryInfo” on page 335
- See “OVsnmpDoRetry” on page 327
- See “OVsnmpRead” on page 345
- See “OVsnmpRecv” on page 347
- See “ovsnmp.conf” on page 304
- See “OVsnmpTrapOpen” on page 352
OVsnmpRead

Purpose

Receives SNMP messages on all active sessions.

Syntax

```
#include <OV\OVsnmp.h>

void OVsnmpRead(struct fd_set *rfdsetp)
```

Description

The OVsnmpRead function is intended to be used with select-to-receive SNMP messages for all active sessions belonging to the calling process. Before calling the OVsnmpRead function, 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, the OVsnmpRead function will not take any action.

The OVsnmpRead function 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 parameter to the OVsnmpOpen function is then called 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.

The ovsnmp library does 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 the OVsnmpFreePdu function 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 the OVsnmpRead function.

- `type`
  The type of PDU that caused the response. The type will be GET_REQ_MSG, GETNEXT_REQ_MSG, or SET_REQ_MSG.

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 daemon has stopped so the SNMP API can no longer receive traps on this session. This exception exists only for sessions created by the OVsnmpTrapOpen function. 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 the OVsnmpOpen or OVsnmpTrapOpen functions.

**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 the OVsnmpOpen or OVsnmpTrapOpen functions. This is application-specific data unique to the session.

**Libraries**
When compiling a program that uses the OVsnmpRead function, link to the following libraries:
- `\usr\ov\lib\nvsnmp.lib`
- `\usr\ov\lib\libov.lib`

**Related Information**
- See "OVsnmpIntro" on page 337.
- See "OVsnmpSend" on page 349.
- See "OVsnmpDoRetry" on page 327.
- See "OVsnmpRecv" on page 347.
- See "OVsnmpGetRetryInfo" on page 335.
- See "OVsnmpOpen" on page 342.
- See "trapd" on page 700.
- See "OVsnmpTrapOpen" on page 352.
**OVsnmpRecv**

**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 field in the OVsnmpPdu structure that is returned. If the value of the error_status field 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 field in the returned PDU.

If an error occurs, the value of the error_status field 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, the OVsnmpRecv function will block until data arrives.

The OVsnmpPdu structure that is returned is created from dynamic memory. It should be freed by a call to the OVsnmpFreePdu or OVsnmpFixPdu functions.

**Parameters**

- `session`  
  Pointer to a valid OVsnmpSession structure returned by a call to the OVsnmpOpen function. This structure contains the socket descriptor and other information needed to receive an SNMP PDU.

**Return Values**

If successful, the OVsnmpRecv function returns a pointer to the PDU that was received. If unsuccessful, it returns NULL.

**Error Codes**

The OVsnmpRecv function returns the error code value in the external variable OVsnmpErrno. If one of the SNMP_SYSERR_* values is returned, 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 daemon has stopped and the SNMP API can no longer receive
traps on this session. This error exists only for sessions created by the
OVsnmpTrapOpen function. The calling process should close the trap
session sock_fd in subsequent calls to select.

Libraries

When compiling a program that uses OVsnmpRecv, link to the following libraries:
- \usr\ov\lib\nvsnmp.lib
- \usr\ov\lib\libov.lib

Related Information
- See "OVsnmpIntro" on page 337
- See "OVsnmpOpen" on page 342
- See "OVsnmpRead" on page 345
- See "OVsnmpSend" on page 349
- See "OVsnmpFixPdu" on page 331
- See "OVsnmpErrString" on page 328
- See "OVsnmpFreePdu" on page 333
OVsnmpSend

Purpose

Sends an SNMP PDU in nonblocking mode.

Related Functions

OVsnmpWSend

Syntax

#include <OV\OVsnmp.h>

int OVsnmpSend(OVsnmpSession *session, OVsnmpPdu *pdu);
int OVsnmpWSend(OVsnmpSession *session, OVsnmpPdu *pdu);

Description

The OVsnmpSend and OVsnmpWSend functions 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 OVsnmpWOpen functions.

Both the OVsnmpSend and OVsnmpWSend functions operate in a nonblocking manner. The specified PDU is encoded and sent without waiting for a response. You should use the OVsnmpBlockingSend function if you want to use the ovsnmp or nvsnmp library in a blocking manner.

The OVsnmpSend and the OVsnmpWSend 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 to be used with select and the OVsnmpGetRetryInfo, OVsnmpDoRetry, and OVsnmpRead functions. 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 call the OVsnmpGetRetryInfo function 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 call the OVsnmpDoRetry function to retransmit the request. If the request arrives before the time-out, the caller should call the OVsnmpRead function to receive the data and pass it to the specified callback function.

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 unnecessary amounts of memory.
Use the OVsnmpWSend routine when the calling process is a Windows application. The calling processes should use the OVsnmpWSend routine only if they are using a windows message loop. Use the OVsnmpWSend routine with the OVsnmpWOpen and OVsnmpWClose routines.

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 OVsnmpWOpen 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 OVsnmpWOpen. The OVsnmpSession structure determines the destination, retry information, callback function pointer, and optionally, a community name, authentication 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, the OVsnmpSend function returns the request ID for the PDU. If unsuccessful, it returns -1 (negative one).

**Error Codes**
The OVsnmpSend function returns the error code value in the external variable 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_SYSERR_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, link to the following libraries:
- `\usr\ov\lib\ovsnmp.lib`
• /usr/ov/lib/libov.lib

Related Information
• See "OVsnmpIntro" on page 337.
• See "OVsnmpOpen" on page 342.
• See "OVsnmpClose" on page 302.
• See "OVsnmpDoRetry" on page 327.
• See "OVsnmpGetRetryInfo" on page 335.
• See "OVsnmpRead" on page 345.
• See "OVsnmpRecv" on page 347.
• See "OVsnmpCreatePdu" on page 325.
• See "OVsnmpAddVarBind" on page 297.
• See "OVsnmpBlockingSend" on page 299.
OVsnmpTrapOpen

Purpose

Connects to the trapd daemon and sets up to receive unfiltered traps.

Related Functions

OVsnmpWTrapOpen

Syntax

```c
#include <OV\OVsnmp.h>
OVsnmpSession *OVsnmpTrapOpen (void (*callback)(), void *callback_data);
#include <OV\OVsnmp.h>
#include <OV\OVsnmpWfns..h>
OVsnmpSession *OVsnmpWTrapOpen ( void (*callback)(), void *callback_data);
```

Dependencies

The OVsnmpTrapOpen function is dependent on the trapd daemon. If the trapd
daemon is not running, the call to the OVsnmpTrapOpen function will fail.

Description

The OVsnmpTrapOpen and OVsnmpWTrapOpen functions 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
daemon and receives all traps that are sent to the trapd daemon. No scoping or
filtering is provided by the OVsnmpTrapOpen function. The new session can be
used with the OVsnmpRead or OVsnmpRecv functions 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 called to process the
inbound trap. It is an error to use OVsnmpRead with a NULL `callback` parameter.

If the calling process will not use callback functions, the OVsnmpRecv function 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 OVsnmpWTrapOpen function with a Windows application. The calling
process should use OVsnmpWTrapOpen only if it is using a Window message loop
or an equivalent function to manage file I/O multiplexing. When a trap arrives, the
callback function supplied in the `callback` parameter will be called by the nvsnmp
library. The nvsnmp library does not perform additional processing.

Parameters

`callback`

A pointer to the routine that will be called to process inbound traps if the
OVsnmpRead function is used to receive the trap. In order for the calling
process to use callback functions, 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.

**Return Values**

If successful, the OVsnmpTrapOpen function returns a pointer to a new OVsnmpSession structure. Memory allocated for the OVsnmpSession structure must be freed by the OVsnmpClose function.

If unsuccessful, the OVsnmpTrapOpen function returns NULL.

**Note:** The trapd daemon might stop after the OVsnmpTrapOpen or OVsnmpWTrapOpen functions return 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**

The OVsnmpTrapOpen function returns the error code value in the external variable OVsnmpErrno. If one of the SNMP_SYSERR_* values is returned, 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, link to the following libraries:

- \usr\ov\lib\nvsnmp.lib
- \usr\ov\lib\libov.lib

**Related Information**

- See [OVsnmpOpen] on page 342
- See [OVsnmpClose] on page 302
- See [trapd] on page 702
- See [OVsnmpRead] on page 345
- See [OVsnmpRecv] on page 347
ovspmd

Purpose

Manages the daemon processes that are part of the NetView program.

Syntax

```
ovspmd [-V] [-f startup_file]
```

Description

The ovspmd daemon manages the daemon processes that are part of the NetView program: starting, stopping, and reporting status on them in response to requests from the user interface commands ovstart, ovstop, and ovstatus. The ovspmd daemon is normally started automatically by ovstart.

The ovstart command requests that the ovspmd daemon start the object manager programs specified in the NetView startup file (SUF). The default startup file is `\usr\ov\conf\ovsuf`. Object managers are configured in a local registration file (LRF) and added to the SUF by the ovaddobj command. If the ovstart command is called with no arguments, the ovspmd daemon will start all object managers configured to run by default (that is, with initial start flag OVs_YES_START in the LRF).

The ovstop command requests the ovspmd command to stop configured object managers. If ovstop is called with no arguments, ovspmd will stop all currently running object managers and exit.

The ovstatus command requests ovspmd command to report the current running status of configured object managers.

Object managers are started by the ovspmd daemon as daemons, in the background, with "" as their working directory, and with stdin, stdout, and stderr attached to \dev\null.

Each object manager can be configured with a dependency list, which is a list of other object managers that must already be running before this object manager can be started successfully. The ovspmd daemon will not start an object manager until all the object managers on which it depends have already successfully initialized.

The ovspmd daemon distinguishes among three classes of object managers:

**OVs_WELL_BEHAVED**

A well-behaved object manager uses the OVsPMD API to communicate with the ovspmd daemon. See [OVsPMD_API on page 356](#) for more information. The object manager sends the ovspmd daemon status information regarding successful and unsuccessful initialization and normal and abnormal termination. The ovspmd daemon determines that a well-behaved object manager has successfully initialized only when it explicitly reports that it has. A well-behaved object manager also exits when it receives the command OVS_CMD_EXIT from the ovspmd daemon.

The status information passed by the object manager to the ovspmd daemon is forwarded to the ovstart or ovstop command, if currently running. The last message received from each object manager is saved and forwarded, on
request, to the ovstatus command. The messages received from well-behaved
object managers are also logged, using the nettl logging and tracing facility, in
the OVS subsystem. Messages indicating normal events, such as successful
initialization, are logged at the INFORMATIVE level.

**OVs NON WELL BEHAVED**
The ovspmd daemon can also manage object managers that do not use the
OVsPMD API (non-well-behaved object managers) only if they do NOT go into
the background of their own accord. See the following information about
OVs_DAEMON. Because it sends no status messages, the ovspmd daemon
considers such an object manager to have initialized successfully if it has not
exited after the time-out configured for it in its LRF.

Non-well-behaved object managers must set up a signal() routine to handle
termination through the SIGTERM or SIGABRT signals if the managers do not
exit within the configured timeout.

**OVs DAEMON**
Object managers that go into the background cannot be managed either with a
communication channel or with signals. The ovspmd daemon can start such an
object manager, but cannot stop or report meaningful status about it, because it
has neither a communication channel nor a process ID for it.

### Flags

- **-V** Run in verbose mode. In this mode, the ovspmd daemon provides detailed
  information about the configuration of object managers which is often more
  information than you need.

- **-f startup_file**
  Read `startup_file` as the startup file (SUF) instead of the default. Note that
  `startup_file` must be an absolute path.

### System Environment

The ovspmd daemon issues error messages regarding configuration errors and
system call failures. These messages are intended to be self-explanatory. If it
currently has an open communication channel with one of the user interface
commands ovstart, ovstop, or ovstatus, the ovspmd daemon sends error messages
through the communication channel to be sent out by the user interface program.

### Libraries

When compiling a program that uses the ovspmd daemon, link to the
\usr\ov\lib\libov.lib library.

### Files

- \usr\ov\conf\ovsuf
- \usr\ov\lrf

### Related Information

- See [ovaddobj](#) on page 277.
- See [ovdelobj](#) on page 279.
- See [ovstart](#) on page 358.
- See [ovstatus](#) on page 360.
- See [ovstop](#) on page 361.
- See [OVsPMD_API](#) on page 356.
- See [lrf](#) on page 48.
OVsPMD_API

Purpose

Describes functions for well-behaved daemon processes in the NetView 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 functions are used by object managers (agents) that must run as background processes in the NetView program in order to be managed by the ovspmd daemon, the process management daemon. An object manager that uses these functions as described is considered well-behaved and should be configured OVs_WELL_BEHAVED in its LRF. See \[lrf on page 48\].

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 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 the ovspmd daemon. A
well-behaved daemon should exit if any of these routines fail.

Libraries

When compiling a program that uses the OVSPMD_API, link to the
\usr\ov\lib\libov.lib library.

Related Information

- See "ovaddobj" on page 277
- See "ovdelobj" on page 279
- See "ovspmd" on page 354
- See "ovstart" on page 358
- See "ovstatus" on page 360
- See "ovstop" on page 361
- See "lrf" on page 48.
ovstart

Purpose

Starts NetView daemon processes.

Syntax

```
ovstart [-v] [-d] [-o ovspmd path] [ovspmd options ...] [--] [object manager names ...]
```

Description

The ovstart command starts the object managers that run as daemon processes, which are part of the NetView program. If called with one or more object_manager_name arguments, the ovstart command starts the designated object managers, after starting the other object managers on which they depend. If called with no arguments, it starts all the object managers that are configured to start by default.

The ovstart command does not exit until all the object managers it has tried to start have either started or definitely failed to start. By default it sends no output unless an object manager fails, but it does produce output if the -v option is used. Running ovstart more than once will not harm the NetView program.

The ovstart command works by sending a start request to the process management daemon, ovspmd. If the ovspmd daemon is not already running, ovstart will start it first.

The ovstart command is not available on clients in a client/server environment.

The object managers are configured by the ovaddobj command from information in local registration files (see "lrf" on page 48). An object manager is named by the first field in the LRF that describes it.

Flags

The ovstart command recognizes the following options. Unrecognized options, arguments beginning with a - symbol, are assumed to be ovspmd options and are added to the command line that is used to start the ovspmd daemon. If the ovspmd daemon is already running, unrecognized options are ignored. The ovstart command is not informed about which options the ovspmd daemon recognizes, nor whether they require arguments. Therefore, if a nonoption (an argument NOT beginning with a - symbol) follows an unrecognized option, it is added to the ovspmd command line.

The first nonoption not immediately following an unrecognized option and succeeding arguments are interpreted as names of object managers to start, and passed to the ovspmd daemon in the start request.

-v  By default, ovstart produces no output unless there are configuration or system errors or the ovspmd daemon or one of the object managers being started fails. In -v (verbose) mode, the ovstart command sends a reassuring status report about each object manager whether it started successfully or not. If the ovspmd daemon was started with its -V option, this status report will be extensive and will include all the information that the ovspmd daemon has about the object manager.
-d In -d (debug) mode, ovstart reports the important stages in its processing, including starting, contacting, and sending the start request to the ovspmd daemon, and the ovspmd daemon’s closing the communication channel.

-o ovspmd_path
This option specifies that the executable for ovspmd is in ovspmd_path instead of in the default location, `usr\ov\bin\ovspmd`. If ovspmd is already running, this option is ignored.

-- Note that if an unrecognized option is followed by a nonoption, the nonoption is ambiguous. It can be intended either as an argument to an option to the ovspmd daemon or as the name of an object manager to start. By default, the ovstart command interprets it as an argument to the unrecognized option. The -- option terminates the options section of the ovstart command line. Arguments following the -- are interpreted as names of object managers to start and are passed to the ovspmd daemon in the start request.

Return Values
If successful, ovstart returns 0 (zero). If unsuccessful, ovstart returns status representing the number of object managers from the start list that were NOT successfully started.

Examples
You can request that the ovspmd daemon start all object managers configured to start by default by entering the following command:

```
ovstart
```

If ovspmd is not already running, ovstart will start it with no options. Only failures will be reported.

System Environment
The ovstart command reports certain command line errors (in particular, too many arguments) and system errors. The messages are prefixed with ovstart and are intended to be self-explanatory. The ovstart command also sends error messages received from the ovspmd daemon. These messages have the ovspmd prefix. The ovstart command does not treat unrecognized options as errors, but the ovspmd daemon does.

Related Information
- See “ovaddobj” on page 277
- See “ovdelobj” on page 279
- See “ovspmd” on page 354
- See “ovstatus” on page 360
- See “ovstop” on page 361
- See “lrf” on page 48
- See “OVsPMD_API” on page 356
ovstatus

Purpose

Reports status of NetView daemon processes.

Syntax

ovstatus [-d] [object_manager_names...]

Description

The ovstatus routine reports the current status of the object managers that run as
daemon processes, which are part of the NetView program. The ovstatus routine
works by sending a status request to the process management daemon, ovspmd. If
called with one or more object_manager_name arguments, it reports status for the
designated object managers. If called with no arguments, it reports the status of all
object managers currently running, including the ovspmd daemon. If the ovspmd
daemon were started with the -V option, this status report will be extensive and will
include all the information that the ovspmd daemon has about the object manager.

Unlike the ovstart command, the ovstatus command will not start the ovspmd
daemon if it is not already running.

The object managers are configured by the ovaddobj command from information in
local registration files (see lrf on page 48). An object manager is named by the first
field in the LRF that describes it.

Flags

The ovstatus routine recognizes the -d option. The first argument that is not an
option and succeeding arguments are interpreted as names of object managers for
which to report status, and are passed to the ovspmd daemon in the status request.

-d In -d (debug) mode, the ovstatus routine reports the important stages in its
processing, including contacting and sending the status request to the ovspmd
daemon, and ovspmd’s closing of the communication channel.

System Environment

The ovstatus routine reports certain command line errors (in particular, too many
arguments) and system errors. The messages have the ovstatus prefix and are
intended to be self-explanatory. The ovstatus routine also sends error messages
received from the ovspmd daemon. These messages have the ovspmd prefix. The
ovstatus routine does not treat unrecognized options as errors, but the ovspmd
daemon does.

Related Information

See ovaddobj on page 277.
See ovdelobj on page 279.
See ovspmd on page 354.
See ovstart on page 358.
See ovstop on page 361.
See lrf on page 48.
ovstop

Purpose

Stops the NetView daemon processes.

Syntax

ovstop [-d] [-v] [object_manager_names ...]

Description

The ovstop command stops the object managers that run as daemon processes, which are part of the NetView program. The ovstop command sends a stop request to the process management daemon, ovspmd. If called with one or more object_manager_name arguments, it stops the designated object managers. If called with no arguments, or if one of the named arguments is OVspMD, it stops all object managers currently running, including the ovspmd daemon.

Unlike the ovstart command, the ovstop command will not start the ovspmd daemon if it is not already running.

The object managers are configured by the ovaddobj command from information in local registration files (see lrf on page 48). An object manager is named by the first field in the LRF that describes it. Unlike the ovstart command, the ovstop command ignores dependency information from the LRF. If other object managers depend on an object manager that is stopped, they must detect that their dependency is no longer met, and exit if appropriate.

Flags

The ovstop command recognizes the options described in the following list. The first argument that is not an option and succeeding arguments are interpreted as names of object managers to stop, and are passed to the ovspmd daemon in the stop request.

- v By default, the ovstop command produces no output unless there are configuration or system errors. In -v (verbose) mode, the ovstop command reports the termination of each object manager as it occurs. If the ovspmd daemon was started with its -V option, this status report will be extensive and will include all the information that the ovspmd daemon has about the object manager.

- d In -d (debug) mode, the ovstop command reports the important stages in its processing, including contacting and sending the stop request to ovspmd, and ovspmd’s closing the communication channel.

System Environment

The ovstop command reports certain command line errors (in particular, too many arguments) and system errors. The messages are prefixed with “ovstop:” and are intended to be self-explanatory. The ovstop command also sends error messages received from the ovspmd daemon. These messages have the ovspmd prefix. The ovstop command does not treat unrecognized options as errors, but the ovspmd daemon does.
Related Information

- See `ovaddobj` on page 277.
- See `ovdelobj` on page 279.
- See `ovsomd` on page 354.
- See `ovstart` on page 358.
- See `ovstatus` on page 360.
- See `lrf` on page 48.
ovtopmd

Purpose
Is the NetView topology manager daemon for IP discovery and layout.

Syntax
ovtopmd [-O]

Description
The ovtopmd daemon is a background process that maintains the ovtopmd topology database used by netmon and ipmap for discovering and laying out IP networks. The ovtopmd process ensures the correctness of the ovtopmd database, and acts as a caching daemon for the other processes involved in IP topology discovery and layout. The ovtopmd daemon is usually started using the ovspmd daemon by request of the ovstart command, and must be started before the netmon daemon and the ipmap command.

The ovwdb process must be executing prior to starting the ovtopmd daemon, and the necessary fields must be initialized in the object database by the -fields option to the ovw command. This is normally handled as part of the installation process. See [OVwRegIntro on page 591] for more information about the ovw command.

Flags
-O Indicates to ovtopmd that it was started by ovspmd, the NetView process management daemon. The ovtopmd process will coordinate and communicate with ovspmd as a member of the OVs_WELL_BEHAVED class. The option should only be used in the LRF file for ovtopmd.

Object Database Fields:
For each type of object represented in the IP topology, there are a number of fields maintained in the OVW object database, as described in Table 11. For more information about NetView field registration, see [OVwRegIntro on page 591].

Table 11. Field Flags Key

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Field</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap</td>
<td>Capability</td>
<td>ovwCapabilityField</td>
</tr>
<tr>
<td>Loc</td>
<td>Locatable</td>
<td>ovwLocateField</td>
</tr>
<tr>
<td>Name</td>
<td>Name field</td>
<td>ovwNameField</td>
</tr>
<tr>
<td>List</td>
<td>List field</td>
<td>ovwListField</td>
</tr>
<tr>
<td>Gen</td>
<td>General field</td>
<td>ovwGeneralField</td>
</tr>
<tr>
<td>&lt;none&gt;</td>
<td>Ordinary field</td>
<td></td>
</tr>
</tbody>
</table>

Table 12. IP Topology Fields

<table>
<thead>
<tr>
<th>Object Type</th>
<th>OVwDb FIELD Name</th>
<th>OVw FieldType</th>
<th>Field Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>“isInterface”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
</tr>
<tr>
<td></td>
<td>“isIP”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
</tr>
<tr>
<td></td>
<td>“IP Address”</td>
<td>ovwStringField</td>
<td>Loc Name</td>
</tr>
</tbody>
</table>
### Table 12. IP Topology Fields (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Type</th>
<th>Location</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>“IP Subnet Mask”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“IP Status”</td>
<td>ovwEnumField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP ifType”</td>
<td>ovwEnumField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP ifPhysAddr”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP ifDescr”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“TopM Network ID”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td>“TopM Segment ID”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td>“TopM Node ID”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Node</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>“isNode”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isIP”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isRouter”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isConnector”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isHub”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isBridge”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isIPRouter”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“isSNMPSupported”</td>
<td>ovwBooleanField</td>
<td>Loc Cap</td>
<td></td>
</tr>
<tr>
<td>“vendor”</td>
<td>ovwEnumField</td>
<td>Cap Gen</td>
<td></td>
</tr>
<tr>
<td>“IP Hostname”</td>
<td>ovwStringField</td>
<td>Loc Name</td>
<td></td>
</tr>
<tr>
<td>“IP Status”</td>
<td>ovwEnumField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP sysDescr”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP sysLocation”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP sysContact”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMP sysObjectID”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“SNMPAgent”</td>
<td>ovwEnumField</td>
<td>Loc Cap Gen</td>
<td></td>
</tr>
<tr>
<td>“TopM Interface List”</td>
<td>ovwStringField</td>
<td>List</td>
<td></td>
</tr>
<tr>
<td>“TopM Interface Count”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Segment</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>“isSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isBusSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isStarSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isTokenRingSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isFDDIRingSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isSerialSegment”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“IP Status”</td>
<td>ovwEnumField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“TopM Interface Count”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td>“TopM Network ID”</td>
<td>owvIntField</td>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>“isNetwork”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“isIP”</td>
<td>ovwBooleanField</td>
<td>Cap</td>
<td></td>
</tr>
<tr>
<td>“IP Network Name”</td>
<td>ovwStringField</td>
<td>Loc Name</td>
<td></td>
</tr>
<tr>
<td>“IP Address”</td>
<td>ovwStringField</td>
<td>Loc Name</td>
<td></td>
</tr>
<tr>
<td>“IP Subnet Mask”</td>
<td>ovwStringField</td>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>“IP Status”</td>
<td>ovwEnumField</td>
<td>Loc</td>
<td></td>
</tr>
</tbody>
</table>
Table 12. IP Topology Fields (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“TopM Default Seg ID”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>“TopM Interface Count”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>“TopM Segment Count”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>TopoInfo</td>
<td>ovwBooleanField</td>
<td>Cap</td>
</tr>
<tr>
<td>“TopM Network Count”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>“TopM Segment Count”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>“TopM Node Count”</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>&amp;rbi</td>
<td>ovwIntField</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

Files

usr\ov\databases\openview\topo\*
ovtopmd database files

\usr\ov\lrfovtopmd.lrf
LRF file for ovtopmd

Related Information

- See “ipmap” on page 37.
- See “netmon” on page 59.
- See “netview” on page 65.
- See “ovspmd” on page 354.
- See “ovstart” on page 358.
- See “ovtopodump” on page 366.
- See “ovwdb” on page 464.
- See “OVwRegIntro” on page 591.
- See “trapd” on page 700.
ovtopodump

Purpose

Prints the contents of the ovtopmd database.

Syntax

ovtopodump [-lLro] [ objects ...]

Description

The ovtopodump command prints out the contents of the ovtopmd database, which is maintained by the ovtopmd daemon. A line is printed for each object containing the object’s type, the object’s name, the object’s ID, which is used by the ovwdb command, the object’s IP address, if any, and the object’s IP status. When the object is an interface, the object ID field contains both the node’s ID and the interface’s ID, separated by a slash (/). If no objects are specified, information is printed about all objects in the ovtopmd database.

Flags

-\texttt{\textbf{-l}} Specifications long form. Instead of one line per object, a line is printed for each field associated with an object, with a blank line between objects. All information maintained in the database for the object is printed.

-\texttt{\textbf{-L}} Includes the Link Level Address for interface objects. This option is not useful if the -l option is used.

-\texttt{\textbf{-o}} Includes the SNMP sysObjectId for node objects. This option is not useful if the -l option is used.

-\texttt{\textbf{-r}} Recursively prints the information based on the IP Topology containment. This option is the default if no options are used, but must be manually specified if the -l option is used. If no object is specified, all networks, segments, and nodes are printed. If a network is specified, all segments and nodes on those segments are printed. If a segment is specified, only nodes on that segment will be printed.

Examples

The following command prints out the global information associated with the ovtopmd topology database.

\texttt{ovtopodump -l}

The output for the above command includes the creation time of the ovtopmd database, the ovwdb object ID associated with the global topology information, and counts of various objects, including the number of networks, segments, nodes, interfaces, and gateways in the IP topology.

The following command prints all information about all objects in the IP topology.

\texttt{ovtopodump -rl}

The following command prints all information about node node1. In this example, the name of the node must be the official IP host name.

\texttt{ovtopodump -l node1}
Files

\usr\ov\databases\openview\topol\*
Map database files and directories

Related Information
See ovtopmd on page 363.
ovtopofix

Purpose

Corrects inconsistencies between ovtopmd and ovwdb.

Syntax

ovtopofix [-achsunv]

ovtopofix [-Cnv]

Description

The ovtopofix command is used to detect and correct inconsistencies that might have developed between the IP topology database maintained by ovtopmd and the database maintained by ovwdb for the ovw command. The default behavior is to remove old hints from the ovwdb database and to verify the managed and removed state of all objects. Hints are objects that contain a selection name and either an IP hostname or IP address, but no other fields. As inconsistencies are discovered, ovtopofix reports actions taken to the standard output.

If updates to the databases are attempted, the ovtopofix command verifies that no ipmap or netmon daemon is currently running.

Flags

- **-a**  Performs all checks and updates that can be performed while ovtopmd is running. Includes the -c, -h, -s, and -u options.
- **-c**  Removes lost IP node objects from the ovwdb database. Lost nodes are objects that contain an IP hostname and other field, but which are not contained in the IP topology database. Implies the -h option.
- **-C**  Attempts compaction of the IP topology databases. This option can be used only if ovtopmd is not running. Compaction of the databases is attempted by reading in all data from the IP topology database, truncating the private IP topology database, and rewriting the data. This operation can be effective in regaining file system space only if a significant number of objects have been deleted from the IP topology database, either through normal user editing or through the use of the ovtopofix command. The -C option is not effective if ovtopmd is started with the -S option.
- **-h**  Removes old hints from the ovwdb database. Hints are objects contained in the ovwdb database that contain a selection name and either an IP hostname or IP address, but no other fields. Netmon had discovered information about these objects during the discovery process, but had not completed adding them to the IP topology database.
- **-n**  Prints the actions that would be taken without updating the ovtopmd or the ovwdb databases. When run with the -n option, ovtopofix does not require stopping ipmap or netmon. This option is useful for checking database consistency without stopping active monitoring of the network.
- **-s**  Verifies the managed and removed state of all objects by comparing the reference counts maintained in the ovwdb database for the ovw command to the state of the object maintained in the IP topology database.
The managed state indicates whether the object is actively monitored by netmon. If an object is managed in any map maintained by the graphical interface, it should be managed in the IP topology database. If an object is not managed in any map, then it should be unmanaged in the IP topology database.

The removed state indicates that an object has been removed from the IP topology database by netmon, but the object should still exist in at least one map maintained by the graphical interface. If an object has been removed from the IP topology database and no longer exists in any maps, the data associated with the object will be deleted from the IP topology database.

-u Updates the reference times on all objects contained in the ovtopmd database. This updating will force ipmap to update the status and state for all objects the next time a map is opened through the graphical interface. This option can be useful if ipmap was unable to complete the synchronization process the last time the ovw command was run on a particular map, or if the status reported in the IP topology database by ovtopodump does not match that displayed through the graphical interface.

-v Specifies verbose mode. In addition to reporting updates, the ovtopofix command reports progress and other actions taken to verify the consistency of the databases.
ovw_config

Purpose

Compiles bitmaps and user-defined symbols.

Syntax

ovw_config

Description

The ovw_config command compiles the bitmaps used for developer and user-defined symbols. Compiling the bitmaps allows them to be read in quickly during startup of the NetView graphical user interface. Run ovw_config whenever you add, delete, or change a bitmap. The ovw_config command will only process symbol registration files.

Files

\usr\ov\bitmaps\*

Bitmap files and directories
ovw_fields

Purpose

Creates the fields defined in the currently installed field registration files.

Syntax

```
ovw_fields
```

Description

The ovw_fields command creates the fields defined in the currently installed field registration files in OVW object database. Run ovw_fields whenever a field registration file is added, deleted, or changed. It will only process field registration files.

Files

```
\usr\ov\fields\*
```

Field registration files and directories
ovw_verify

Purpose
Checks the syntax for all the installed application, symbol type, and field registration files.

Syntax
ovw_verify

Description
The ovw_verify command checks the syntax for all the installed application, symbol type, and field registration files. Verifies the semantics of the application registration files and reports conflicts between application registration information. Only registration files are processed.

Files

\usr\ov\registration\*
  Application registration files and directories
\usr\ov\fields\*
  Field registration files and directories
\usr\ov\symbols\*
  Symbol registration files and directories
OVwAckMapClose

Purpose

Acknowledges a map close event.

Syntax

```c
#include <OV\ovw.h>

int OVwAckMapClose(OVwMapInfo *map, time_t close_time);
```

Description

OVwAckMapClose must be called in response to the ovwMapClose event. (See "OVwMapCloseCB" on page 583.) It informs the NetView program that the application has received the ovwMapClose event and has completed any changes needed before the NetView program closes the map.

Note: Failure to call OVwAckMapClose in response to the ovwMapClose event will cause the graphical interface to wait for two minutes 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 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 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 GUI API has not been initialized with OVwInit.
Examples

The following example illustrates the use of OVwAckMapClose in an OVwMapCloseCB callback:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwAckMapClose, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See [Netview](#) on page 65.
- See [OVwError](#) on page 520.
- See [OVwGetMapInfo](#) on page 544.
- See [OVwInit](#) on page 564.
- See [OVwMapCloseCB](#) on page 583.
- See [OVwApiIntro](#) on page 395.


OVwAckUserSubmapCreate

Purpose

Acknowledges a user, submap-create event.

Syntax

```c
#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" on page 650). 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 that does not have a child submap through the graphical interface.

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 program was lost.

- **[OVw_MAP_NOT_OPEN]**
  The argument map does not specify an open map.
Examples

The following example illustrates the use of OVwAckUserSubmapCreate in an OVwUserSubmapCreateCB callback:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwAckUserSubmapCreate, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "netview" on page 65.
- See "OVwAddCallback" on page 382.
- See "OVwCreateSubmap" on page 443.
- See "OVwInit" on page 564.
- See "OVwUserSubmapCreateCB" on page 650.
- See "OVwApiIntro" on page 395.
- See "OVwGetMapInfo" on page 544.
- See "OVwError" on page 520.
**OVwAddActionCallback**

**Purpose**

Registers a callback for a registered action.

**Related Functions**

OVwRemoveActionCallback

**Syntax**

```c
#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 arg[0]

- **callbackProc**
  Specifies a procedure to be called when a user activates the specified menu item or executable symbol.

- **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.

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 in
   OVwError. 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 program was lost.

   [OVw_OUT_OF_MEMORY]
      A memory allocation failure occurred.

   [OVw_OVW_NOT_INITIALIZED]
      The GUI API has not been started with OVwInit.

Implementation Specifics
   OVwAddActionCallback supports single-byte and multibyte character code sets.

Libraries
   When compiling a program that uses OVwAddActionCallback and
   OVwRemoveActionCallback, link to the following libraries:
      • /usr/ov/lib/libovw.lib
      • /usr/ov/lib/libov.lib
      • /usr/ov/lib/libntl.lib

Related Information
   • See "netview" on page 65.
   • See "OVwError" on page 520.
   • See "OVwInit" on page 564.
   • See "OVwApiIntro" on page 395.
   • See "OVwGetMapInfo" on page 544.
OVwAddAlertCallback

Purpose

Registers handlers of NetView alerts.

Related Functions

OVwRemoveAlertCallback

Syntax

```c
#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 alert messages that other applications have generated through OVwAlertMsg.

Description

OVwAddAlertCallback adds an application callback procedure for handling classes of NetView 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.

Notes:

1. Use OVwAddAlertCallback and OVwRemoveAlertCallback only if the sole purpose of your application is to dispatch NetView alert messages.
2. 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 an NetView 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 program was lost.

   [OVw_OUT_OF_MEMORY]
      A memory allocation failure occurred.

   [OVw_OVW_NOT_INITIALIZED]
      The GUI API has not been initialized with OVwInit.

Examples
   The following example is an application that displays NetView 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) {
         perror(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 multibyte character code sets.

**Libraries**

When compiling a program that uses OVwAddAlertCallback or OVwRemoveAlertCallback, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

**Related Information**

- See "netview" on page 65
- See "OVwAlertMsg" on page 393
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwAddCallback

Purpose

Registers procedures to process NetView events.

Related Functions

OVwRemoveCallback

Syntax

```c
#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 events of an object with the specified capabilities.

OVwRemoveCallback removes a previously added callback procedure for a specified NetView event and set of capabilities.

Calls to OVwAddCallback for differing sets of capabilities are cumulative; multiple procedures can be registered for each NetView event with each event processing objects with different capabilities. Only one callback can be registered for a particular capabilitySet.

Parameters

- **event**
  Specifies the NetView 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 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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Implementation Specifics

OVwAddCallback supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwAddCallback or OVwRemoveCallback, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libnt1.lib

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwAddHelpCallback

Purpose

Registers a handler for application help requests.

Related Functions

OVwRemoveHelpCallback

Syntax

```c
#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 help system.

Description

OVwAddHelpCallback adds an application callback procedure for handling NetView 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.

Notes:
1. Use OVwAddHelpCallback and OVwRemoveHelpCallback only if the sole purpose of your application is to implement the NetView help system.
2. 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI API has not been initialized with OVwInit.

### Implementation Specifics
OVwAddHelpCallback supports single-byte and multibyte character code sets.

### Libraries
When compiling a program that uses OVwAddHelpCallback or OVwRemoveHelpCallback, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

### Related Information
- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwShowHelp" on page 646
- See "OVwApiIntro" on page 395
OVwAddInput

Purpose

Adds an event source.

Related Functions

OVwRemoveInput

Syntax

```c
#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 event processing mechanism as another source of events. When the specified condition occurs on the fileDescriptor, as detected by select, proc is called and is passed the parameter userData.

OVwRemoveInput removes an input event, created by a previous call to OVwAddInput, from the NetView event processing mechanism.

Parameters

- **fileDescriptor**
  Specifies the source file descriptor to be added to the NetView event processing loop.

- **conditionMask**
  Specifies a mask that might contain the ovwReadMask flag, indicating 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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwAddInput supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwAddInput or OVwRemoveInput, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwAddMenuItem

Purpose

Adds a menu item to a menu,

Related Functions

OVwRemoveMenuItem

Syntax

```c
#include <OV/ovw.h>
#include <OV/ovw_reg.h>

int OVwAddMenuItem(char *menuId, char **menuItemId);
int OVwRemoveMenuItem(char *menuId, char **menuItemId);
```

Description

OVwAddMenuItem ties a menu item to a registered menu or the graphical interface menu bar in the current registration context. See OVwGetRegContext on page 552 for information about changing the registration context.

OVwRemoveMenuItem removes a menu item that is tied to 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 graphical interface 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 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 which points to a pointer to a menu item ID returned from the OVwFindMenuItem call. 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 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 GUI 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 graphical interface menu bar that does not contain menu functions.

Implementation Specifics

OVwAddMenuItem supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwAddMenuItem or OVwRemoveMenuItem, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See [“netview” on page 63](#)
- See [“OVwError” on page 520](#)
- See [“OVwInit” on page 564](#)
- See [“OVwLockRegUpdates” on page 579](#)
- See [“OVwCreateMenuItem” on page 446](#)
- See [“OVwSaveRegUpdates” on page 617](#)
- See [“OVwApiIntro” on page 395](#)
- See [“OVwRegIntro” on page 591](#)
- See [“OVwGetRegContext” on page 552](#)
- See [“OVwAddMenuItemFunction” on page 390](#)
**OVwAddMenuItemFunction**

**Purpose**

Adds a menu item function to a menu item.

**Related Functions**

OVwRemoveMenuItemFunction

**Syntax**

```c
#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 for "OVwGetRegContext" on page 552 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 on page 388), 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 sessions.

**Parameters**

*menuItemId*

Specifies a pointer to a menu item ID returned from an OVwFindMenuItem call.

*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 notepad \winnt\system32\drivers\etc\hosts, this is equivalent to specifying !notepad \winnt\system32\drivers\etc\hosts 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 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 GUI 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 multibyte character code sets are supported.
Libraries

When compiling a program that uses OVwAddMenuItemFunction or OVwRemoveMenuItemFunction, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Files

ov\ovw.h

Related Information

- See “netview” on page 65
- See “OVwAddMenuItem” on page 388
- See “OVwError” on page 520
- See “OVwInit” on page 564
- See “OVwLockRegUpdates” on page 579
- See “OVwCreateMenu” on page 444
- See “OVwSaveRegUpdates” on page 617
- See “OVwApilntro” on page 395
- See “OVwRegIntro” on page 591
- See “OVwGetMapInfo” on page 544
OVwAlertMsg

Purpose
Issues an NetView 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 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 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI API has not been initialized with OVwInit.

**Implementation Specifics**

OVwAlertMsg supports single-byte and multibyte character code sets.

**Libraries**

When compiling a program that uses OVwAlertMsg, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

**Files**

ov\ovw.h

**Related Information**

- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApilntro" on page 395.
- See "OVwAddAlertCallback" on page 373.
OVwApiIntro

Purpose

Provides an overview of the graphical user interface API.

Syntax

cc < or CC or ld > ... -lovw -lov -lntl

Description

The GUI API contains a registration facility and a set of library routines that enable applications to integrate with the NetView program, the NetView network, and the systems management user interface. The ovw command provides the graphical user 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 GUI 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 about registration files. See page 697. 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, map routines, and map editing.

Concepts:

- The following concepts are important for understanding the GUI API. Refer to the Tivoli NetView for Windows 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, and describe.
- Filtering GUI 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 unique 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 and 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\reg\advanced\c, \usr\ov\reg\beginner\c, and \usr\ov\registration\c directories. 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\fields\c 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\symbols\c 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, 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 on page 591](#) for more information about 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 13.

<table>
<thead>
<tr>
<th>Type of Return Code</th>
<th>Code for Success</th>
<th>Code for Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer value</td>
<td>0 (zero)</td>
<td>-1</td>
</tr>
<tr>
<td>Pointer</td>
<td>Valid pointer</td>
<td>NULL</td>
</tr>
<tr>
<td>ID values</td>
<td>Valid ID</td>
<td>NULL ID</td>
</tr>
</tbody>
</table>

**Note:** The macro OVwIsIdNull should be used to check for a NULL ID value.

The error codes for the GUI API are listed in the `<OV\ovw_errs.h>` header file.

The following routines are error handling routines:

**OVwError**
Returns the last GUI 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 GUI API. OVwInit establishes communication with the NetView 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 object database routines (those routines beginning with OVwDb) but does not need to use the rest of the GUI API or to be started from the NetView program.

The following routines are process management routines:
OVwInit
Initializes a connection with the NetView program

OVwDbInit
Initializes a connection with the ovwdb daemon process

OVwDone
Terminates the connection with the NetView program

**Event Processing:** An application can register to receive various asynchronous events from the NetView 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 GUI 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 on page 585](#) for an example of registering for an GUI API event. The GUI API events and the corresponding callbacks are defined in [OVwEventIntro on page 522](#).

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 GUI API events and actions:

**OVwAddActionCallback**
Add a callback for an action request

**OVwAddCallback**
Add a callback for an GUI API event

**OVwRemoveActionCallback**
Remove a callback for an action request

**OVwRemoveCallback**
Remove a callback for an GUI API event

For events to be received from the NetView 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 messages can call OVwInitApiEvents. This call will enable an application to get both NetView events and messages.
- An application that does not need to process messages can call OVwMainLoop to receive NetView events. Use OVwAddInput to include additional input sources in the event processing loop.
- An application that needs to perform select processing can use OVwFileDescriptor and OVwProcessEvent to get NetView events. Most applications do not need to use these low-level routines.

The following routines are event processing routines:

**OVwAddInput**
Adds a callback for input on a file descriptor for use with OVwMainLoop
OVwFileDescriptor
Gets the NetView input file descriptor

OVwInitApiEvents
Initializes application for receiving of NetView callbacks

OVwMainLoop
Continuously processes NetView events

OVwPeekInputEvent
Determines whether a particular input event is pending

OVwPeekOVwEvent
Determines whether a particular NetView event is pending

OVwPending
Determines whether any NetView event is pending

OVwProcessEvent
Processes the next NetView event

OVwRemoveInput
Removes a callback for input on a file descriptor for use with OVwMainLoop

General Routines: The following routines are general routines:

OVwAlertMsg
Generates an alert message

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

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 OVwDbCreateField routine. The OVwDbCreateField routine is not needed for fields registered through a field registration file.

The following routines are field routines:

OVwDbAppendEnumConstants
Appends name constants for an enumerated type

OVwDbCreateField
Creates a new field
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

**OVwDbCreateObjectByHostname**
Creates a graphical interface object with an IP hostname

**OVwDbCreateObjectBySelectionName**
Creates a graphical interface object with a object name

**OVwDbDeleteObject**
Deletes a graphical interface object

The following routines are object routines that access fields:

**OVwDbFreeFieldBindList**
Frees a field value list

**OVwDbFreeFieldValue**
Frees a field value

**OVwDbFreeObjectFieldList**
Frees an object field list
OVwDbFreeObjectIdList
Frees an object ID list

OVwDbGetCapabilityFieldValues
Gets a list of capability field values for an object

OVwDbGetFieldValue
Gets a particular field value for an object

OVwDbGetFieldValues
Gets a list of all field values for an object

OVwDbGetFieldValuesByObjects
Gets the value of a field for each object

OVwDbGetNameFieldValues
Gets a list of name field values for an object

OVwDbGetUniqObjectName
Gets a unique value for a name field

OVwDbHostnameToObjectIId
Converts IP hostname to object ID

OVwDbListObjectsByFieldValue
Locates all objects with a field value

OVwDbListObjectsByFieldValues
Locates all objects with field values

OVwDbNameToObjectIId
Converts a name field value to object ID

OVwDbObjectIIdToHostname
Converts object ID to IP hostname

OVwDbObjectIIdToSelectionName
Converts object ID to object name

OVwDbSelectionNameToObjectIId
Converts object name to object ID

OVwDbSetFieldValue
Sets the value of a field for an object

OVwDbSetHostname
Sets the IP hostname of an object

OVwDbSetSelectionName
Sets the object name 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

Symbol Type Routines: 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. The following routines are symbol type routines:

OVwListSymbolTypeCaps
Gets the default capabilities for a symbol type
**OVwListSymbolTypes**  
Gets a list of all registered symbol types

**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 GUI API events that an application can receive to monitor changes in the map. See "OVwEventIntro" on page 523 for a complete list.

The following map routines operate only on an open map:

**OVwAckMapClose**  
Acknowledges a map close event

**OVwBeginMapSync**  
Begins the map synchronization phase

**OVwCopyMapInfo**  
Copies map information

**OVwEndMapSync**  
Ends the map synchronization phase

**OVwFreeMapInfo**  
Frees map information

**OVwGetAppConfigValues**  
Gets application configuration values for a map

**OVwGetMapInfo**  
Gets map information

**OVwSetAppConfigValues**  
Sets application configuration values for a map

The following routines are submap routines:

**OVwClearBackgroundGraphic**  
Clears the background graphic for a submap

**OVwCreateSubmap**  
Creates a submap

**OVwDeleteSubmap**  
Deletes a submap

**OVwDisplaySubmap**  
Displays a submap

**OVwFreeSubmapInfo**  
Frees submap information
OVwFreeSubmapList
Frees a list of submaps

OVwGetSubmapInfo
Gets submap information

OVwListSubmaps
Gets a list of submaps

OVwSetBackgroundGraphic
Sets the background graphic for a submap

OVwSetSubmapName
Sets the name of a submap

The following routines are the symbol routines:

OVwClearSymbolApp
Clears application interest in a symbol

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

OVwCreateSymbol
Creates a symbol

OVwCreateSymbolByHostname
Creates a symbol with an IP hostname

OVwCreateSymbolByName
Creates a symbol with an object name

OVwCreateSymbolBySelectionName
Creates a symbol with a object name

OVwCreateSymbols
Creates multiple symbols

OVwDeleteSymbol
Deletes a symbol

OVwDeleteSymbols
Deletes multiple symbols

OVwFreeSymbolInfo
Frees symbol information

OVwFreeSymbolList
Frees a symbol list

OVwGetConnSymbol
Gets a connection symbol

OVwGetSymbolInfo
Gets symbol information
OVwGetSymbolsByObject
Gets a list of symbols representing an object

OVwListSymbols
Gets a list of symbols on a submap

OVwSetStatusOnSymbol
Sets the status of a symbol

OVwSetStatusOnSymbols
Sets the status of multiple symbols

OVwSetSymbolApp
Expresses application interest in a symbol

OVwSetSymbolBehavior
Makes a symbol explodable or executable

OVwSetSymbolLabel
Sets the label of a symbol

OVwSetSymbolPosition
Moves a symbol

OVwSetSymbolStatusSource
Sets the status source of a symbol

OVwSetSymbolType
Changes the symbol type of a symbol

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:

OVwFreeObjectInfo
Frees map-specific object information

OVwFreeObjectList
Frees an object list

OVwGetObjectInfo
Gets map-specific object information

OVwListObjectsOnMap
Gets a list of objects on the open map

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" on page 522 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 program generates a query event, such as ovwQueryAddSymbol.
2. An application responds by calling a verification routine, such as OVwVerifyAdd.
3. The NetView program generates a final confirm event, such as ovwConfirmAddSymbol, to notify the application of the operation results. See "OVwVerifyAdd" on page 652 for more details about this interaction.

The following routines are map editing verification routines:

**OVwVerifyAdd**
Verifies user add of an object

**OVwVerifyAppConfigChange**
Verifies user change of application configuration values

**OVwVerifyConnect**
Verifies user connect of two symbols

**OVwVerifyDeleteSymbol**
Verifies user delete of symbols

**OVwVerifyDescribeChange**
Verifies user change of object description information

Header Files:

*<OV\ovw.h>*
This header file is the main header file for the GUI API. It defines most of the GUI API structures and routines. It includes the following header files:
- *
- *
- *
- *

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 GUI 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 GUI API, such as OVwBoolean and OVwStatusType. It also defines the macros OVwIsIdEqual and OVwIsIdNull for testing and comparing IDs.
This header file defines string constants for predefined symbol types registered in certain symbol type registration files in the `\usr\ov\symbols\c` directory.

## Files

- `<ov\ovw.h>`
  - Header file for the GUI API
- `<ov\ovw_errs.h>`
  - Header file for the GUI 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 GUI 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 GUI API, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

## Related Information

- See "netview" on page 65
- See "OVwAckMapClose" on page 373
- See "OVwAckUserSubmapCreate" on page 375
- See "OVwAddActionCallback" on page 377
OVwBeginMapSync

Purpose

Begins map synchronization phase.

Related Functions

OVwEndMapSync

Syntax

```c
#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 GUI API has not been initialized with OVwInit.

Examples
The following example illustrates the use of OVwBeginMapSync and OVwEndMapSync in an OVwMapOpenCB callback:

```c
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 multibyte character

Libraries
When compiling a program that uses OVwBeginMapSync or OVwEndMapSync, link to the following libraries:
- `/usr/ov/lib/ovw.lib`
- `/usr/ov/lib/ov.lib`
- `/usr/ov/lib/ntl.lib`

Related Information
- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwGetMapInfo" on page 544
- See "OVwInit" on page 564
- See "OVwMapOpenCB" on page 585
- See "OVwApiIntro" on page 395
- See "OVwGetMapInfo" on page 544
OVwCheckAction

Purpose

Enables applications to check the validity of the NetView 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 an application action (the calling application or a different one) is applicable to the specified selection list.

OVwDoAction invokes an application action on the specified selection list.

These APIs permit applications to trigger NetView 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 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 action callback. A NULL `submap` is permitted if it is not critical to the functions of the application action.
Return Values

If the application action is valid for the specified selections, map, and submap, OVwCheckAction returns TRUE. If the application action is not valid or available, it returns FALSE.

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 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 GUI 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 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwCheckAction or OVwDoAction, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libnltl.lib

Related Information

- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
- See "OVwGetMapInfo" on page 544.
OVwConfirmCapabilityChangeCB

Purpose

Functions as a callback for an object capability change event.

Syntax

```c
#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 capability fields are set for an object when the symbol type is set for a symbol representing the object. See "OVwApiIntro" on page 395 for an overview of the GUI API including the role of the asynchronous NetView 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 user 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" on page 652), the Describe Object editing operation (see "OVwVerifyDescribeChange" on page 663), and direct updates (see "OVwDbSetFieldValue" on page 508). 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 owMapOpen 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 owConfirmCapabilityChange. 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, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libntl.lib

Related Information
• See "netview" on page 65.
• See "OVwAddCallback" on page 382.
• See "OVwCreateSymbol" on page 453.
• See "OVwDbSetFieldValue" on page 505.
• See "OVwSetSymbolType" on page 643.
• See "OVwVerifyAdd" on page 652.
• See "OVwVerifyDescribeChange" on page 668.
• See "OVwApiIntro" on page 395.
• See "OVwRegIntro" on page 591.
• See "OVwGetMapInfo" on page 544.
OVwConfirmCreateObjectsCB

Purpose

Functions as a callback for a create object event.

Syntax

```c
#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 on page 393” for an overview of the GUI API including the role of the asynchronous NetView 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, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See “Netview” on page 63
- See “OVwAddCallback” on page 382
• See [OVwApiIntro on page 395].
OVwConfirmCreateSubmapsCB

Purpose

Functions as a callback for a create-submap event.

Syntax

```c
#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 program will generate the OVwConfirmCreateSubmaps event when the user or an application creates a submap. See "OVwApiIntro" on page 395 for an overview of the GUI API including the role of the asynchronous NetView 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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libntlib.lib

Related Information

- See "OVwGetMapInfo" on page 544.
- See "netview" on page 65.
- See "OVwAddCallback" on page 382.
• See "OVwApiIntro" on page 395.
OVwConfirmCreateSymbolsCB

Purpose

Functions as a callback for a create symbol event.

Syntax

```c
#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 program will generate the ovwConfirmCreateSymbols event when the user or an application creates a symbol. See “OVwApiIntro” on page 395 for an overview of the GUI API including the role of the asynchronous NetView events.

To be notified when a symbol is created on the open map, an 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. The `map` parameter can be obtained using OVwGetMapInfo or saved from the OVwMapOpen event using OVwCopyMapInfo.

- `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, link to the following libraries:

- `\us\ov\lib\libovw.lib`
- `\us\ov\lib\libov.lib`
- `\us\ov\lib\libnt.lib`

Related Information

- See “OVwGetMapInfo” on page 544.
- See “netview” on page 63.
• See "OVwAddCallback" on page 383.
• See "OVwApiIntro" on page 395.
OVwConfirmDeleteObjectsCB

Purpose

Functions as a callback for a delete object event.

Syntax

```c
#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 on page 395" for an overview of the GUI API, including the role of the asynchronous NetView 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 user 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 user 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 should update its current structures to reflect this deletion. For ovwAllMapsScope, however, the routine should 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 on page 473" for more information.

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.

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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "netview" on page 66.
- See "OVwAddCallback" on page 383.
- See OVwConfirmDeleteSymbolsCB in "OVwVerifyDeleteSymbol" on page 665.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
- See "OVwGetMapInfo" on page 544.
- See "OVwDbDeleteObject" on page 473.
OVwConfirmDeleteSubmapsCB

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 a submap is deleted from the map. An submap is deleted from the map when the last symbol of the submap is deleted. See "OVwApiIntro on page 395" for an overview of the GUI API including the role of the asynchronous NetView events.

For an application to be notified when a submap is deleted from the open map, the application should register this callback using the OVwAddCallback function call and ovwConfirmDeleteSubmaps as the event type.

The NetView 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.

- **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, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`
Related Information

- See "netview" on page 65.
- See "OVwAddCallback" on page 382.
- See "OVwApiIntro" on page 395.
OVwConfirmHideSymbolsCB

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" on page 399 for an overview of the 
GUI API including the role of the asynchronous NetView events.

To be notified when an object is hidden or unhidden, an 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 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, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "OVwGetMapInfo" on page 544.
- See "netview" on page 65.
- See "OVwAddCallback" on page 383.
- See "OVwApiIntro" on page 395."
OVwConfirmManageObjectsCB

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 been registered to them when a manage or unmanage operation is selected by the user. See [OVwApiIntro on page 393](#) for an overview of the GUI API including the role of the asynchronous NetView events.

To be notified when an object is managed or unmanaged, an application 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 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 unmanaged 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 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 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:

```c
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`, link to the following libraries:

- `/usr/ov/lib/lobovw.lib`
- `/usr/ov/lib/lobov.lib`
- `/usr/ov/lib/libntl.lib`

**Related Information**

- See ["OVwGetMapInfo" on page 544](#)
- See ["netview" on page 65](#)
- See ["OVwAddCallback" on page 383](#)
- See ["OVwApiIntro" on page 395](#)
**OVwConfirmMoveSymbolCB**

**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" on page 395 for an overview of the GUI API including the role of the asynchronous NetView 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 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" on page 635 for more details.

The ovwConfirmMoveSymbol event is not sent to applications when the user moves a symbol from one submap to another; the NetView 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 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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "netview" on page 65.
- See "OVwAddCallback" on page 382.
- See "OVwSetSymbolPosition" on page 636.
- See "OVwApiIntro" on page 395.
OVwConfirmObjectStatusCB

Purpose

Functions as a callback for a change objects status event.

Related Functions

OVwConfirmSymbolStatusCB
OVwConfirmCompoundStatusCB

Syntax

```c
#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,
oveConfirmSymbolStatusChange 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 propagated
status of an object is changed. The propagated status is the status propagated to
the object from symbols in its child submap according to the selected propagation
rules. The propagated 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 propagated 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 propagated status of its associated object, or a
direct change in the symbol status through the OVwSetStatusOnSymbol or
OVwSetStatusOnSymbols.

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 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, link to the following libraries:
   - \usr\ov\lib\libovw.lib
   - \usr\ov\lib\libov.lib
   - \usr\ov\lib\libntl.lib

Related Information
   - See “OVwGetMapInfo” on page 544.
   - See “netview” on page 65.
   - See “OVwAddCallback” on page 383.
   - See “OVwSetStatusOnObject” on page 623.
   - See OVwSetStatusOnSymbols in “OVwSetStatusOnObject” on page 623.
   - See “OVwSetSymbolStatusSource” on page 640.
   - See “OVwApiIntro” on page 395.
OVwCountObjectsByFieldId

Purpose

Returns the number of objects that have this fieldID set.

Syntax

OVwCountObjectsByFieldId

Description

This is an efficient way to count various objects.
OVwCreateAction

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);
void OVwFreeActionRegInfo(OVwActionRegInfo *actionInfo);
int OVwSetAction(char *actionId, OVwActionRegInfo *actionInfo);
OVwActionRegInfo *OVwGetAction(char *actionId);

Description
OVwCreateAction creates the specified action in the current registration context.

OVwDeleteAction deletes 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.

OVwSetAction modifies registration information for the specified action in the current registration context.

OVwGetAction retrieves registration information for the specified action in the current registration context.

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 will become permanent only after calling OVwSaveRegUpdates.

Use these functions 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 the 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 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 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, NULL is returned.

**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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  There is not enough memory to store the callback registration information.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI API has not been started with OVwInit.

- **[OVw_PERMISSION_DENIED]**
  OwLockRegUpdates was not called prior to calling this function.

**Implementation Specifics**

OVwCreateAction and its related functions support single-byte and multibyte character code sets.

**Files**

OV\ovw_reg.h

**Libraries**

When compiling a program that uses OVwCreateAction and one of its related functions, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- `/usr/ov/lib/libntl.lib`

Related Information
- See "`netview`" on page 63.
- See "`OVwError`" on page 520.
- See "`OVwInit`" on page 564.
- See "`OVwLockRegUpdates`" on page 579.
- See "`OVwSaveRegUpdates`" on page 617.
- See "`OVwApiIntro`" on page 395.
- See "`OVwRegIntro`" on page 591."
**OVwCreateApp**

**Purpose**
Manipulates NetView application registration information.

**Related Functions**
- OVwDeleteApp
- OVwGetApp
- OVwSetApp
- OVwFreeAppRegInfo

**Syntax**
```c
#include <OV\ovw.h>
#include <OV\ovw_reg.h>

int OVwCreateApp(char *appName, OVwAppRegInfo *appInfo);
int OVwDeleteApp(char *appName);
OVwAppRegInfo *OVwGetApp();
int OVwSetApp(OVwAppRegInfo *appInfo);
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 in the current registration context.

OVwSetApp modifies registration information for the application that is in 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 will become permanent only after calling OVwSaveRegUpdates.

Use these functions 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 application.
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:

```c
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 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 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 a NetView 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  
There is not enough memory to store the callback registration information.

- **[OVw_OVW_NOT_INITIALIZED]**
  
The GUI API has not been initialized with OVwInit.

- **[OVw_PERMISSION_DENIED]**
  
You have not called OVwLockRegUpdates prior to calling this function.

**Implementation Specifics**

OVwCreateApp or one of its related functions supports single-byte and multibyte character code sets.

**Libraries**

When compiling a program that uses OVwCreateApp or one of its related functions, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`
OVwCreateMenu

Purpose

Creates and deletes menu registration.

Related Functions

OVwDeleteMenu

Syntax

```c
#include <OV\ovw.h>
#include <OV\ovw_reg.h>

int OVwCreateMenu(char *menuId);
int OVwDeleteMenu(char *menuId);
```

Description

OVwCreateMenu creates a menu in the current registration context. When 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 cannot contain menu items or 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 will 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 program was lost.

- **[OVw_MENU_EXISTS]**
  The specified `menuId` is already registered in the current registration context.
The specified menu *menuId* is still referenced by other menu items and cannot be deleted.

The menu still contains menu items and cannot be deleted.

The specified *menuId* is not registered in the current registration context.

There is not enough memory to store the callback registration information.

The GUI API has not been initialized with OVwInit.

You have not called OVwLockRegUpdates prior to calling this function.

**Implementation Specifics**

OVwCreateMenu and OVwDeleteMenu support single-byte and multibyte character code sets.

**Libraries**

When compiling a program that uses OVwCreateMenu and OVwDeleteMenu, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libnlt.lib`

**Related Information**

- See "netview" on page 63
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwLockRegUpdates" on page 579
- See "OVwSaveRegUpdates" on page 617
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See "OVwAddMenuItem" on page 386
OVwCreateMenuItem

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);
OVwMenuItemRegInfo *OVwGetMenuItem(char *menuItemId);
int OVwSetMenuItem(char *menuItemId, OVwMenuItemRegInfo *menuItemInfo);
void OVwFreeMenuItemRegInfo(OVwMenuItemRegInfo *menuItemInfo);

Dependencies
OVwCreateMenuItem creates a menu item in the current registration context.

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 will 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:
```
```c
typedef struct {
  char *label;
  char *mnemonic;
  char *accelerator;
  int precedence;
} OVwMenuItemRegInfo;
```

The members of this structure are as follows:

```
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, NULL is returned.

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 program was lost.
```

```
[OVw_MENUITEM_EXISTS]
The specified menuItemId is already registered in the current registration
context.
```
The specified menu item, $menuId$, is still included in a menu and cannot be deleted.

The menu item still contains menu item functions and cannot be deleted.

The specified $menuItemId$ is not registered in the current registration context.

There is not enough memory to store the callback registration information.

The GUI API has not been initialized with OVwInit.

You have not called OVwLockRegUpdates prior to calling this function.

The specified precedence is not within the valid range of precedence values.

**Implementation Specifics**

OVwCreateMenuItem and its related functions support single-byte and multibyte character code sets.

**Libraries**

When compiling a program that uses an OVwMenuItem function, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

**Files**

OV\ovw_reg.h

**Related Information**

- See [OVwCreateMenuitem](#) on page 446
- See [OVwFindMenuitem](#) on page 525
- See [netview](#) on page 65
- See [OVwError](#) on page 520
- See [OVwInit](#) on page 564
- See [OVwLockRegUpdates](#) on page 573
- See [OVwSaveRegUpdates](#) on page 617
- See [OVwApiIntro](#) on page 395
- See [OVwRegIntro](#) on page 591
**OVwCreateSubmap**

**Purpose**

Creates a submap.

**Related Functions**

OVwDeleteSubmap

**Syntax**

```c
#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" on page 652). 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 properties, and deleting the submap.

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 user 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" on page 453.

Creating a submap does not cause the submap to be displayed. Because the user can display submaps through the graphical user 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.

**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 can be saved from the ovwMapOpen event using OVwCopyMapInfo.
**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 set the error code value that OVwError returns. The following list describes the possible errors:

**[OVw_CONNECTION_LOST]**
- The connection to the NetView 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 GUI 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.

Implementation Specifics
OVwCreateSubmap supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses an OVwCreateSubmap function, link to the following libraries:
• `/usr/ov/lib/libovw.lib`
• `/usr/ov/lib/libov.lib`
• `/usr/ov/lib/libntl.lib`

Files
OV\ovw.h

Related Information
• See [netview](#) on page 65
• See [OVwConfirmDeleteObjectsCB](#) on page 423
• See [OVwConfirmDeleteSubmapsCB](#) on page 425
• See [OVwCreateSymbol](#) on page 453
• See [OVwDisplaySubmap](#) on page 515
• See [OVwError](#) on page 520
• See [OVwGetMapInfo](#) on page 544
• See [OVwInit](#) on page 564
• See [OVwIsIdNull](#) on page 567
• See [OVwSetSymbolApp](#) on page 628
• See [OVwVerifyAdd](#) on page 653
• See [OVwVerifyDeleteSymbol](#) on page 665
• See [OVwApiIntro](#) on page 395
**OVwCreateSymbol**

**Purpose**

Creates symbols.

**Related Functions**

- OVwCreateSymbols
- OVwCreateSymbolByName
- OVwCreateSymbolBySelectionName
- OVwCreateSymbolByHostname
- OVwCreateComponentSymbol
- OVwCreateComponentSymbolByName
- OVwCreateConnSymbol
- OVwCreateConnSymbolByName
- OVwDeleteSymbol
- OVwDeleteSymbols

**Syntax**

```c
#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,
```
Description

OVwCreateSymbol and its related functions create and delete symbols on a submap of the open map. OVwCreateSymbol 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" on page 449).

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" on page 585 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 \texttt{OVwSetSymbolPosition} on page \pageref{OVwSetSymbolPosition} 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 \texttt{OVwSetSymbolPosition}.

\texttt{OVwCreateSymbols} creates multiple symbols with a single call. This is more efficient than making individual calls to create each symbol. \texttt{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 \texttt{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, \texttt{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 \texttt{OVwSymbolCreateInfo} structure will indicate which list elements failed. Also upon return, the symbol_id field of the \texttt{OVwSymbolCreateInfo} structure will contain the symbol ID of all those symbols that were successfully created.

\texttt{OVwCreateSymbolByName} creates a symbol representing the object having the name field value indicated by \texttt{name} and adds it to the submap of the open map identified by \texttt{submapId}. If an object with the specified name does not exist, it will be automatically created using \texttt{OVwDbCreateObject}. If \texttt{name} is NULL, an object will be automatically created with a system-generated name.

\texttt{OVwCreateSymbolBySelectionName} creates a symbol representing the object identified by the specified object name. If an object with the specified object name does not exist, it will be automatically created using \texttt{OVwDbCreateObjectBySelectionName}.

\texttt{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 \texttt{OVwDbCreateObjectByHostname}.

\texttt{OVwCreateComponentSymbol} creates a symbol representing the object identified by \texttt{objectId} on the child submap of a compound object identified by \texttt{parentId}. If the child submap of the object \texttt{parentId} does not exist, it is automatically created using \texttt{OVwCreateSubmap} with the submap policy, \texttt{ovwSharedSubmap}.

\texttt{OVwCreateComponentSymbolByName} creates a symbol representing the object, which is identified by \texttt{name} on the child submap of a compound object, which is identified by \texttt{parentld}. If the child submap of the object \texttt{parentld} does not exist, it is automatically created using \texttt{OVwCreateSubmap} with the submap policy \texttt{ovwSharedSubmap}. If an object with the specified \texttt{name} does not exist, it will be automatically created using \texttt{OVwDbCreateObject}.

\texttt{OVwCreateConnSymbol} creates a connection symbol representing an object identified by \texttt{objectId} between two icon symbols that are identified by \texttt{endpoint1} and \texttt{endpoint2} on the submap that is identified by \texttt{submapId}. If the layout of the submap on which the symbol is being created (\texttt{ovwBusLayout}, or \texttt{ovwRingLayout}) has a backbone, one of the endpoints can be specified as \texttt{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 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 endpoints 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" on page 443).

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 endpoint.

`endpoint2`
- Specifies the symbol ID of an icon symbol that is a connection endpoint. 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.

`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 object 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 the optional symbol position information structure. This parameter will normally be NULL, which uses the automatic layout algorithm for the submap. See [OVwSetSymbolPosition on page 635](#) 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 program was lost.

[OVw_DB_CONNECTION_LOST]
The connection to the ovwdb objec database 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 GUI 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 parentld does not exist.

[OVw_SUBMAP_NOT_FOUND]
The submap specified by submapld 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 symbolld. 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:
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.

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.

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.

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:

Both connection endpoints have the value ovwNullSymbolId.

Both connection endpoints are the same.

One of the connection endpoint symbols does not exist on the open map.

The variety of a connection endpoint symbol is valid. Only icon symbols are allowed as connection endpoints.

The two connection endpoints are on different submaps. Both connection endpoints must be on the same submap.

One of the endpoint name style fields of the OVwSymbolCreateInfo structure (for OVwCreateSymbols) has a value that is not valid.

An attempt was made to create a connection directly on a metaconnection submap.

A value of ovwSubmapBackbone is specified as a connection endpoint for a submap that does not have a backbone.

The OVwDeleteSymbol and OVwDeleteSymbols routines return the following additional error:

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 OVwCreateSymbol or one of its related functions, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Files

- OV\ovw.h
- OV\ovw_types.h

Related Information

- See OVwDbCreateObject” on page 471.
- See OVwDbUnsetFieldValue” on page 513.
- See OVwIsIdNull” on page 567.
- See netview” on page 65.
- See ovwdb” on page 464.
- See OVwConfirmCreateSymbolsCB” on page 421.
- See OVwConfirmCreateObjectsCB” on page 417.
- See OVwConfirmCreateSubmapsCB” on page 419.
- See OVwConfirmDeleteSubmapsCB” on page 423.
- See OVwConfirmDeleteObjectsCB” on page 423.
- See OVwCreateSubmap” on page 443.
- See OVwDbCreateObject” on page 471.
- See OVwDbDeleteObject” on page 473.
- See OVwDbUnsetFieldValue” on page 513.
- See OVwError” on page 520.
- See OVwGetMapInfo” on page 544.
- See OVwInit” on page 564.
- See OVwIsIdNull” on page 567.
- See OVwMapOpenCB” on page 585.
- See OVwSetSymbolPosition” on page 633.
- See OVwVerifyAdd” on page 652.
- See OVwVerifyDeleteSymbol” on page 665.
- See OVwApiIntro” on page 395.
**OVwDatabaseStillAlive**

**Purpose**
Verifies that ovwdb is running.

**Description**
This routine verifies that ovwd is running. It can be used by any application that uses ovwdb API calls.

**Return Values**

```c
BOOL OVwDatabaseStillAlive() // true==still alive
```
ovwdb

Purpose
Maintains the NetView object database.

Syntax
ovwdb [-O] [-n number_of_object]

Description
The ovwdb command starts a background process that maintains the NetView graphical user interface object database used by the NetView program and applications that use the GUI API. Access to the NetView program is provided by the OVwDb API. In order to access the data managed by the OVW object database, an application must establish communication with the ovwdb process. To do this, an application must be integrated with the NetView program (the application has made the OVwInit call). The OVwDbInit call must be used if the application is to function without being part of NetView graphical user interface.

Flags

-O Indicates to ovwdb that it was started by ovspmd, the NetView Network Management Server process management daemon. The ovwdb process will coordinate and communicate with ovspmd as a member of the OVs_WELL_BEHAVED class. Use this option only in the LRF file for ovwdb.

-n number_of_objects
Controls the number of objects maintained in the cache. The ovwdb command acts as a caching daemon for the object information stored in the OVW object database. The ovwdb.lrf file has this option set to a default value of 5000 objects. However, this value can be changed as conditions require. For more information about changing the value set for this option and to set criteria for determining a correct new value, refer to the Tivoli NetView for Windows Programmer’s Guide.

Libraries
When compiling a program that uses ovwdb, link to the following libraries:
- /usr/ov/lib/libovw.lib
- /usr/ov/lib/libov.lib
- /usr/ov/lib/libntl.lib

Files

/usr/ov/databases/openview/ovwdb/*
The NetView object database files
/usr/ov/lrf/ovwdb.lrf
LRF file for ovwdb

Related Information
- See ovstart* on page 358
- See ovspmd* on page 354
- See netview* on page 65
- See OVwInit* on page 564
• See "OVwDbInit" on page 494.
• See "lrf" on page 48.
OVwDbAppendEnumConstants

Purpose
Append 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" on page 506 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 GUI 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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "ovwdb" on page 464
- See "OVwDbCreateField" on page 468
- See OVwDbGetFieldEnumByName in "OVwDbGetFieldValue" on page 482
- See OVwDbGetFieldEnumByValue in "OVwDbGetFieldValue" on page 482
- See "OVwDbInit" on page 494
- See "OVwDbSetEnumConstants" on page 506
- See OVwDbSetFieldEnumByName in "OVwDbSetFieldValue" on page 508
- See OVwDbSetFieldEnumByValue in "OVwDbSetFieldValue" on page 508
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwDbCreateField

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 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 program that this field can be used to locate
  objects for the user interface. The field name is displayed in the Find
dialog box when this flag is set. This flag can be used with fields of any
data type.

- **ovwGeneralField**
  Provides the ability to have nonapplication-specific fields that are not
displayed in any application-specific dialog box displayed in the Property Page 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 fieldID 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 the ovwdb object database was lost.

[OVw_DB_NOT_INITIALIZED]
The GUI 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 fieldID 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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libnfl.lib

**Files**

OV\ovw_obj.h

**Related Information**

- See [ovwdb](#) on page 464.
- See [OVwDbFieldNameToFieldId](#) on page 475.
- See [OVwDbGetFieldInfo](#) on page 480.
- See [OVwDbGetFieldValue](#) on page 482.
- See [OVwDbInit](#) on page 494.
- See [OVwDbListFields](#) on page 496.
- See [OVwDbSetFieldValue](#) on page 508.
- See [OVwError](#) on page 520.
- See [OVwInit](#) on page 564.
- See OVwIsIdEqual in [OVwIsIdNull](#) on page 567.
- See [OVwIsIdNull](#) on page 567.
- See [OVwApiIntro](#) on page 395.
**OVwDbCreateObject**

**Purpose**

Creates an object in the OVW object database.

**Related Functions**

- OVwDbCreateObjectBySelectionName
- OVwDbCreateObjectByHostname

**Syntax**

```c
#include <OV\ovw_obj.h>

OVwObjectId OVwDbCreateObject(OVwFieldBinding *name);

OVwObjectId OVwDbCreateObjectBySelectionName(char *selectionName);

OVwObjectId OVwDbCreateObjectByHostname(char *hostname);
```

**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 object name field for the object, a object name is automatically created and set for the object. An error results if the object name is provided but is not unique. If NULL is provided as input, OVwDbCreateObjectBySelectionName creates an object, and sets the object name to a system-generated name.

OVwDbCreateObjectBySelectionName creates a new object in the OVW Object database. An error occurs if an object exists with the selection name. If NULL is provided as input, OVwDbCreateObjectBySelectionName creates an object and sets the object name to a system-generated name.

OVwDbCreateObjectByHostname creates a new object in the OVW Object database. The object 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.

**Parameters**

- **hostname**
  Specifies a pointer to the host name to 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.

- **selectionName**
  Specifies a pointer to the object name of the object. The selectionName must be unique for all objects.

**Return Values**

If successful, OVwDbCreate, 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 object IDs.
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 the ovwdb object database was lost.

[OVw_DB_NOT_INITIALIZED]
The GUI 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_OBJECT_NULL_NAME]
The name provided to the function is NULL.

Implementation Specifics

OVwDbCreateObject and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbCreateObject or one of its related functions, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "ovwdb" on page 464.
- See "OVwDbGetUniqObjectName" on page 490.
- See "OVwDbInit" on page 494.
- See "OVwDbSetFieldValue" on page 503.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwIsId Null" on page 567.
- See "OVwApiIntro" on page 395.
- See "OVwDbUnsetFieldValue" on page 513.
OVwDbDeleteObject

Purpose

Deletes an object from the OVW object database.

Syntax

```c
#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 object 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.

Parameters

`objId`

Specifies the object ID of the object to be deleted.

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 the ovwdb object database was lost.

- **[OVw_DB_NOT_INITIALIZED]**
  The GUI 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:

```c
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.

```c
OVwDbDeleteObject(object_id);
```

Implementation Specifics

OVwDbDeleteObject supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbDeleteObject, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libnt1.lib`

Related Information

- See [OVwdb](#) on page 464.
- See [OVwDbCreateField](#) on page 468.
- See [OVwDbCreateObject](#) on page 471.
- See [OVwDbInit](#) on page 494.
- See [OVwDbSetFieldValue](#) on page 508.
- See [OVwDbUnsetFieldValue](#) on page 513.
- See [OVwError](#) on page 520.
- See [OVwInit](#) on page 564.
- See [OVwApiIntro](#) on page 395.
OVwDbFieldNameToFieldId

**Purpose**

Converts a field name to a field ID.

**Related Functions**

OVwDbFieldIdToFieldName

**Syntax**

```c
#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 a 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](on page 468) 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 the ovwdb object database was lost.
[OVw_DB_NOT_INITIALIZED]
The GUI 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 *fieldname* or the provided *fieldID* 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, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
- See "OVwDb" on page 464.
- See "OVwDbCreateField" on page 468.
- See "OVwDbGetFieldInfo" on page 480.
- See "OVwDbInit" on page 494.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwIsIdNull" on page 567.
- See "OVwApiIntro" on page 395.
**OVwDbGetEnumConstants**

**Purpose**

Accesses enumerated type values.

**Related Functions**

- OVwDbFreeEnumConstants
- OVwDbGetEnumValue
- OVwDbGetEnumName

**Syntax**

```c
#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 value 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 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 the ovwdb database was lost.

**[OVw_DB_NOT_INITIALIZED]**
The GUI API has not been initialized with OVwInit.

**[OVw_DB_OPEN_FAILED]**
An attempted connection to the NetView 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 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, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

**Related Information**

- See [ovwdb](#) on page 464.
- See [OVwDbCreateField](#) on page 468.
- See OVwDbGetFieldEnumByValue in [OVwDbGetFieldValue](#) on page 482.
- See OVwDbGetFieldEnumByName in [OVwDbGetFieldValue](#) on page 483.
• See "OVwDbInit" on page 494.
• See "OVwDbSetEnumConstants" on page 506.
• See OVwDbSetFieldEnumByName in "OVwDbSetFieldValue" on page 508.
• See OVwDbSetFieldEnumByValue in "OVwDbSetFieldValue" on page 508.
• See "OVwError" on page 520.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
OVwDbGetFieldInfo

Purpose

Returns information about a database field.

Related Functions

OVwDbFreeFieldInfo

Syntax

```c
#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](#) on page 468.

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 the ovwdb object database was lost.
- **[OVw_DB_NOT_INITIALIZED]**
  - The GUI 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 multibyte character code sets.

Libraries
When compiling a program that uses OVwDbGetFieldInfo or OVwDbFreeFieldInfo, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libntl.lib

Related Information
• See "ovwdb" on page 464
• See "OVwDbCreateField" on page 468
• See OVwDbFieldIdToFieldName in "OVwDbFieldNameToFieldId" on page 475
• See "OVwDbFieldNameToFieldId" on page 475
• See "OVwDbInit" on page 494
• See "OVwInit" on page 564
• See "OVwApiIntro" on page 395
• See "OVwError" on page 520
**OVwDbGetFieldValue**

**Purpose**

Obtains field values for an object.

**Related Functions**

- OVwDbGetFieldEnumByValue
- OVwDbGetFieldEnumByName
- OVwDbGetFieldBooleanValue
- OVwDbGetFieldIntegerValue
- OVwDbGetFieldStringValue
- OVwDbFreeFieldValue

**Syntax**

```c
#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. The string is returned in static memory. 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 FALSE; 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, NULL is returned. 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 the ovwdb object database was lost.

- **[OVw_DB_NOT_INITIALIZED]**
  The GUI 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 multibyte character code sets.

**Libraries**

When compiling a program that uses an `OVwDbGetFieldValue` function, link to the following libraries:
• \usr\olv\lib\libovw.lib
• \usr\olv\lib\libov.lib
• \usr\olv\lib\libntl.lib

Related Information

• See "ovwrlib" on page 464.
• See "OVwDbGetFieldValues" on page 486.
• See "OVwDbGetFieldValuesByObjects" on page 488.
• See "OVwDbInit" on page 494.
• See OVwDbSetFieldBooleanValue in "OVwDbSetFieldValue" on page 508.
• See OVwDbSetFieldStringValue in "OVwDbSetFieldValue" on page 508.
• See "OVwDbSetFieldValue" on page 508.
• See "OVwError" on page 520.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
OVwDbGetFieldValues

Purpose

Obtains 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 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 the ovwdb object database was lost.

[OVw_DB_NOT_INITIALIZED]
The GUI 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 objectld does not exist.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

Implementation Specifics

OVwDbGetFieldValues and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses an OVwDbGetFieldValues function, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See “ovwdb” on page 464.
- See “OVwDbGetFieldValuesByObjects” on page 488.
- See “OVwDbInit” on page 494.
- See OVwDbSetFieldBooleanValue in “OVwDbSetFieldValue” on page 508.
- See OVwDbSetFieldStringValue in “OVwDbSetFieldValue” on page 508.
- See “OVwDbSetFieldValue” on page 508.
- See “OVwError” on page 520.
- See “OVwInit” on page 564.
- See “OVwApiIntro” on page 395.
**OVwDbGetFieldValuesByObjects**

**Purpose**

Obtains values for a field for multiple objects.

**Related Functions**

OVwDbFreeObjectFieldList

**Syntax**

```c
#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

- `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, NULL is returned.

**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 the ovwdb object database was lost.

- `[OVw_DB_NOT_INITIALIZED]`
  
  The GUI 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 multibyte character code sets.

### Libraries

When compiling a program that uses OVwDbGetFieldValuesByObjects or OVwDbFreeObjectFieldList, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

### Related Information

- See "ovwrdb" on page 464
- See "OVwDbGetFieldValue" on page 482
- See "OVwDbInit" on page 494.
- See OVwDbSetFieldBooleanValue in "OVwDbSetFieldValue" on page 508.
- See OVwDbSetFieldStringValue in "OVwDbSetFieldValue" on page 508.
- See "OVwDbSetFieldValue" on page 508.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
OVwDbGetUniqObjectName

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, an NetView 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 the ovwdb object database was lost.

- **[OVw_DB_NOT_INITIALIZED]**
  The GUI 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 namefieldld does not exist.

[OVw_FIELD_NOT_NAME]  The namefieldld 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwDbGetUniqObjectName, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "ovwbh" on page 464
- See "OVwDbCreateField" on page 468
- See "OVwDbInit" on page 494
- See "OVwDbNameToObjectld" on page 502
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwDbHostnameToObjectíd

Purpose
Converts an IP host name to an object ID.

Related Functions
OVwDbObjectIdToHostname

Syntax
#include <OV\ovw_obj.h>

OVwObjectId OVwDbHostnameToObjectíd(char *hostname);
char *OVwDbObjectIdToHostname(OVwObjectId objectId);

Description
The OVW object database defines the IP Hostname field. 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.

OVwDbHostnameToObjectíd 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, OVwDbHostnameToObjectíd 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
OVwDbHostnameToObjectíd and OVwDbObjectIdToHostname set the error code
value that OVwError returns. The following list describes the possible errors:

[OVw_DB_CONNECTION_LOST]
The connection to the ovwdb object database was lost.
[OVw_DB_NOT_INITIALIZED]
The GUI 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.

OVwDbHostnameToObjectIId returns the following error:

[OVw_FIELD_VALUE_NULL]
The hostname provided is a NULL character pointer.

Implementation Specifics
OVwDbHostNameToObjectIId and OVwObjectIdToHostName support single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwDbHostnameToObjectIId or OVwDbObjectIdToHostName, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
- See "ovwdb" on page 464
- See "OVwDbCreateObject" on page 471
- See "OVwDbInit" on page 494
- See "OVwDbSetFieldValue" on page 508
- See "OVwDbNameToObjectIId" on page 502
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwIsIdNull" on page 567
- See OVwIsIdEqual in "OVwIsIdNull" on page 567.
- See "OVwApiIntro" on page 395
OVwDbInit

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 an NetView application and the NetView object 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 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_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 the ovwdb object database.

- **[OVw_CONNECTION_LOST]**
  The connection to the ovwdb object database was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

Implementation Specifics
OVwDbInit supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwDbInit, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntlib.lib`

Related Information
- See "NetView" on page 65.
- See "Ovwdb" on page 464.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
OVwDbListFields

Purpose

Lists OVW object database fields.

Related Functions

OVwDbFreeFieldList

Syntax

```c
#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" on page 468.

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. This is the logical OR of the following values:
  - **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.

- **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 the ovwdb object database was lost.

**[OVw_DB_NOT_INITIALIZED]**
The GUI 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]**
An invalid combination of search filters 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:

```c
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:

```c
OVwFieldList *fieldList;
if((fieldList=OVwDbListFields(ovwAllFields)= =NULL) {
    printf("Error: %d.\n",OVwError());
}
```

**Implementation Specifics**

OVwDbListFields and OVwDbFreeFieldList support single-byte and multibyte character code sets.

**Libraries**

When compiling a program that uses OVwDbListFields or OVwDbFreeFieldList, link to the following libraries:
- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`
- `/usr/ov/lib/libntlib.lib`

**Related Information**
- See "ovwdb" on page 464
- See "OVwDbCreateField" on page 468
- See "OVwDbGetFieldInfo" on page 480
• See "OVwDbInit" on page 494.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
• See "OVwError" on page 520.
**OVwDbListObjectsByFieldValue**

**Purpose**

Lists objects by field value.

**Related Functions**

OVwDbListObjectsByFieldValues

OVwDbFreeObjectIdList

**Syntax**

```c
#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.

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.

These functions take one argument and the argument cannot be NULL.

**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 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 GUI 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwDbListObjectsByFieldValue or one of its related functions, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "ovwdb" on page 464
- See "OVwDbCreateField" on page 468
- See "OVwDbCreateObject" on page 471
- See "OVwDbInit" on page 494
- See "OVwDbSetFieldValue" on page 503
- See "OVwError" on page 520
- See "OVwInit" on page 564
• See "OVwApiIntro" on page 395.
OVwDbNameToObjectId

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" on page 468.

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 for testing and comparing object ID values.

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 the ovwdb object database was lost.

[OVw_DB_NOT_INITIALIZED]
The GUI 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
OVwDbNameToObjectld supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwDbNameToObjectld, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
- See "ovwdb" on page 464
- See “OVwDbCreateField” on page 468
- See “OVwDbInit” on page 494
- See “OVwDbSetFieldValue” on page 508
- See “OVwInit” on page 564
- See “OVwIsIdNull” on page 567
- See OVwIsIdEqual in “OVwIsIdNull” on page 567
- See “OVwApiIntro” on page 395
OVwDbSelectionNameToObjectId

Purpose
Converting an object ID to a object 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 object name. The object name for an object is defined at the time the object is created. See [OVwDbCreateObject on page 471] for more information regarding the creation of objects. OVwDbSelectionNameToObjectId and OVwDbObjectIdToSelectionName provide a convenient method of converting an object name to an object ID and an object ID to a object name.

OVwDbSelectionNameToObjectId returns the OVwObjectId of the object that has as a object name matching the value provided by selectionName.

OVwDbObjectIdToSelectionName returns the object name defined for the object identified by objectId.

Parameters

selectionName
Specifies a pointer to the object 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 object 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 object name defined for the object identified by objectId. If unsuccessful, NULL is returned. 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 the ovwdb object database was lost.

[OVw_DB_NOT_INITIALIZED]
The GUI 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 objectld or selectionName was found.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

OVwDbSelectionNameToObjectld returns the following error:

[OVw_FIELD_VALUE_NULL]
The selectionName provided is a NULL character pointer.

Implementation Specifics
OVwDbSelectionNameToObjectld and OVwObjectIdToSelectionName support single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwDbSelectionNameToObjectld or OVwObjectIdToSelectionName, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information
- See [ovwdb] on page 464
- See [OVwDbCreateObject] on page 471
- See [OVwDbInit] on page 494
- See [OVwError] on page 520
- See [OVwInit] on page 564
- See [OVwIsIdNull] on page 567
- See OVwIsIdEqual in [OVwIsIdNull] on page 567
- See [OVwApiIntro] on page 395
OVwDbSetEnumConstants

**Purpose**

Sets values for an enumerated data type.

**Syntax**

```c
#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" on page 468 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 keyword Unset. If enumConstants does not include this keyword 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 the ovwdb object database was lost.
- `[OVw_DB_NOT_INITIALIZED]` The GUI 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.
Implementation Specifics

OVwDbSetEnumConstants supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDbSetEnumConstants, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "OVwDb" on page 464
- See "OVwDbAppendEnumConstants" on page 466
- See "OVwDbCreateField" on page 468
- See OVwDbGetFieldEnumByValue in "OVwDbGetFieldValue" on page 482
- See OVwDbGetFieldEnumByName in "OVwDbGetFieldValue" on page 483
- See "OVwDbInit" on page 494
- See OVwDbSetFieldEnumByName in "OVwDbSetFieldValue" on page 508
- See OVwDbSetFieldEnumByValue in "OVwDbSetFieldValue" on page 508
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
OVwDbSetFieldValue

Purpose

Sets a field value for an object.

Related Functions

OVwDbSetFieldBooleanValue
OVwDbSetFieldEnumByValue
OVwDbSetFieldEnumByName
OVwDbSetFieldIntegerValue
OVwDbSetFieldStringValue

Syntax

```
#include <OV\ovw_obj.h>

int OVwDbSetFieldValue(OVwObjectId objectId, OVwFieldId fieldId, OVwFieldValue *fieldValue);

int OVwDbSetFieldBooleanValue(OVwObjectId objectId, OVwFieldId fieldId, OVwBoolean booleanValue);

int OVwDbSetFieldEnumByValue(OVwObjectId objectId, OVwFieldId fieldId, OVwEnumField enumFields);

int OVwDbSetFieldEnumByName(OVwObjectId objectId, OVwFieldId fieldId, char *name);

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.

OVwDbSetFieldBooleanValue sets a value for a field of type ovwBooleanField.

OVwDbSetFieldEnumByValue and OVwDbSetFieldEnumByName 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" on page 508 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

- **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 the ovwdb object database was lost.

- **[OVw_DB_NOT_INITIALIZED]**
  The GUI 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 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:
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 multibyte character code sets.

Libraries

When compiling a program that uses an OVwDbSetFieldValue function, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "OVwdb" on page 464.
- See OVwDbGetFieldEnumByName in "OVwDbGetFieldValue" on page 482.
- See OVwDbGetFieldEnumByValue in "OVwDbGetFieldValue" on page 482.
- See "OVwDbGetFieldInfo" on page 480.
- See "OVwDbInit" on page 494.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwDbSetEnumConstants" on page 506.
- See "OVwError" on page 520.
OVwDbSetSelectionName

Purpose
Sets the object name.

Related Functions
OVwDbSetHostname

Syntax

```c
#include <OV.ovw_obj.h>

int OVwDbSetSelectionName(OVwObjectId objectId, char *sname);
int OVwDbSetHostname(OVwObjectId objectId, char *hname);
```

Description
The object name and the IP host name fields are defined by the NetView 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 "OVwDbNameToObjectIds" on page 502. These functions provide a convenient way to change values assigned to these fields for a given object.

OVwDbSetSelectionName resets the value of the object 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 object 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 the ovwdb object database was lost.

- **[OVw_DB_NOT_INITIALIZED]**
  The GUI API has not been initialized with OVwInit.
[OVw_DB_OPENFAILED]
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
multibyte character code sets.

Libraries
When compiling a program that uses OVwDbSetSelectionName or
OVwDbSetHostname, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libntlib.lib

Related Information
• See "ovwdb" on page 464.
• See "OVwDbGetUniqObjectName" on page 490.
• See "OVwDbHostnameToObjectID" on page 492.
• See "OVwDbInit" on page 494.
• See "OVwDbNameToObjectID" on page 503.
• See "OVwDbSelectionNameToObjectID" on page 504.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
• See "OVwError" on page 520.
**OVwDbUnsetValue**

**Purpose**

Unsets field values for an object.

**Related Functions**

OVwDbUnsetValue

**Syntax**

```cpp
#include <OV\ovw_obj.h>

int OVwDbUnsetValue(OVwObjectId objectId, OVwFieldId fieldId);
int OVwDbUnsetValue(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 OVwDbGetField with the parameters `objectId` and `fieldId` (or any field ID listed in `fieldIdList`) will result in an error `OVw_FIELD_VALUE_NULL`. These calls do not affect the field definition. See [*OVwDbCreateField*](#) on page 468 for more information on field definitions.

OVwDbUnsetValue provides the ability to remove a field value for an object.

OVwDbUnsetValue 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, OVwDbUnsetValue and OVwDbUnsetValue return 1. If unsuccessful, they return 0 (zero).

OVwDbUnsetValue 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`.

**Error Codes**

OVwDbUnsetValue and OVwDbUnsetValue set the error code value that OVwError returns. The following list describes the possible errors:

- **[OVw_DB_CONNECTION_LOST]**
  
  The connection to the ovwdb object database was lost.
[OVw_DB_NOT_INITIALIZED]
The GUI 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 Object Name field, the OVW
Maps Exists field, and the OVW Maps Managed field.

Implementation Specifics
OVwDbUnsetFieldValue supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwDbUnsetFieldValue or
OVwDbUnsetFieldValues, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libntl.lib

Related Information
• See OVwdb* on page 464.
• See "OVwDbInit" on page 494.
• See "OVwInit" on page 564.
• See OVwApiIntro on page 395.
• See "OVwDbGetFieldValue" on page 482.
• See "OVwDbCreateField" on page 468.
• See "OVwError" on page 520.
OVwDisplaySubmap

Purpose

Displays a submap.

Syntax

```c
#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 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 GUI 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 multibyte character code sets.
Libraries

When compiling a program that uses OVwDisplaySubmap, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "netview" on page 65.
- See "OVwCreateSubmap" on page 449.
- See "OVwError" on page 520.
- See "OVwGetMapInfo" on page 544.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwGetMapInfo" on page 544.
OVwDone

Purpose

Terminates an application’s connection to the NetView program.

Syntax

```c
#include <OV\ovw.h>

Void OVwDone();
```

Description

OVwDone terminates the communications between an application and the NetView program. After calling OVwDone, the application may not make any NetView calls, except to OVwError and OVwErrorMsg. OVwDone should be called immediately before an NetView application exits.

Return Values

The OVwDone routine does not return a value.

Error Codes

OVwDone sets the error code value that OVwError returns. The possible error is:

[OVw_OVW_NOT_INITIALIZED]

The GUI API has not been initialized with OVwInit.

Implementation Specifics

OVwDone supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwDone, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See [netview](#) on page 65
- See [OVwInit](#) on page 564
- See [OVwError](#) on page 520
- See [OVwApiIntro](#) on page 395
- See [OVwErrorMsg](#) on page 521
OVwEndSessionCB

Purpose

Functions as a callback for an end-of-session message.

Syntax

```c
#include <OV\ovw.h>

void (*OVwEndSessionCB) (void *userData, OVwEventType type, 
                        OVwBoolean normalEnd);
```

Description

To receive a message indicating that the NetView session is exiting, use OVwAddCallback to register a callback function of type OVwEndSessionCB to be called when an ovwEndSession message is generated.

**Note:** It is recommended that every application register to receive the ovwEndSession message so that all applications can terminate correctly when the NetView program exits.

Parameters

normalEnd

Indicates whether the end of the NetView 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 program received.

Examples

The following code fragment shows an example of registering a callback routine for receiving an end session message:

```c
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 multibyte character code sets.
Libraries

When compiling a program that uses OVwEndSessionCB, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "netview" on page 63
- See "OVwAddCallback" on page 382
- See "OVwAPIIntro" on page 395
OVwError

Purpose
Returns the error code set by the last GUI API call.

Syntax
#include <OV\ovw.h>

int OVwError();

Description
OVwError returns the error value set by the previous GUI API call. It can be tested immediately after a failed NetView call to determine the exact reason for the failure.

Examples
The following code fragment illustrates the way OVwError can be used with OVwErrorMsg:

```c
if (OVwInit() < 0) {
    fprintf(stderr, "OVwInit: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}
```

Implementation Specifics
OVwError supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwError, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
- See "netview" on page 65.
- See "OVwErrorMsg" on page 521.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
OVwErrorMsg

Purpose

Returns a textual description of an GUI API error value.

Syntax

```c
#include <OV\ovw.h>

char *OVwErrorMsg(int error);
```

Description

OVwErrorMsg maps an GUI API error value to a string that contains text describing the meaning of the specified error value.

Parameters

`error`

A GUI API error value, 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:

```c
if (OVwInit() < 0) {
    fprintf(stderr, "OVwInit: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}
```

Implementation Specifics

OVwErrorMsg supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwErrorMsg, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntlf.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwApiIntro" on page 395
OVwEventIntro

Purpose

Provides an introduction to NetView events.

Description

An application can be registered to receive various asynchronous events from the NetView 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 GUI API events, such as when the NetView 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" on page 585 for an example of registering for a GUI API event. The GUI API events and the corresponding callbacks are defined in the header file <OV\ovw.h>. In addition to Table 14, see "OVwApiIntro" on page 395 for a complete overview of the GUI API.

Table 14. GUI API Events and Their Callbacks

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ovwEndSession</td>
<td>NetView session termination</td>
<td>OVwEndSessionCB</td>
</tr>
<tr>
<td>ovwSelectListChange</td>
<td>Map selection list changed</td>
<td>OVwSelectListChangeCB</td>
</tr>
<tr>
<td>ovwMapOpen</td>
<td>Map open</td>
<td>OVwMapOpenCB</td>
</tr>
<tr>
<td>ovwMapClose</td>
<td>Map close</td>
<td>OVwMapCloseCB, OVwAckMapClose</td>
</tr>
<tr>
<td>ovwQueryAppConfigChange</td>
<td>Application configuration change query</td>
<td>OVwVerifyAppConfigChange</td>
</tr>
<tr>
<td>ovwConfirmAppConfigChange</td>
<td>Application configuration changed</td>
<td>OVwVerifyAppConfigChange</td>
</tr>
<tr>
<td>ovwQueryDescribeChange</td>
<td>Description change query</td>
<td>OVwVerifyDescribeChange</td>
</tr>
<tr>
<td>ovwConfirmDescribeChange</td>
<td>Description changed</td>
<td>OVwVerifyDescribeChange</td>
</tr>
<tr>
<td>ovwQueryAddSymbol</td>
<td>Query to add symbol to map</td>
<td>OVwVerifyAdd</td>
</tr>
<tr>
<td>ovwConfirmAddSymbol</td>
<td>Symbol added to map</td>
<td>OVwVerifyAdd</td>
</tr>
<tr>
<td>ovwQueryConnectSymbols</td>
<td>Query to connect objects</td>
<td>OVwVerifyConnect</td>
</tr>
<tr>
<td>ovwConfirmConnectSymbols</td>
<td>Objects connected</td>
<td>OVwVerifyConnect</td>
</tr>
<tr>
<td>ovwQueryDeleteSymbols</td>
<td>Query to delete symbols</td>
<td>OVwVerifyDeleteSymbol</td>
</tr>
<tr>
<td>ovwConfirmDeleteSymbols</td>
<td>Symbols deleted from map</td>
<td>OVwVerifyDeleteSymbol</td>
</tr>
<tr>
<td>ovwConfirmDeleteObjects</td>
<td>Objects deleted from map</td>
<td>OVwConfirmDeleteObjectsCB</td>
</tr>
<tr>
<td>ovwConfirmDeleteSubmaps</td>
<td>Submaps deleted from map</td>
<td>OVwConfirmDeleteSubmapsCB</td>
</tr>
</tbody>
</table>
### Table 14. GUI API Events and Their Callbacks (continued)

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ovwConfirmCreateSymbols</td>
<td>Symbols created on map</td>
<td>OVwConfirmCreateSymbolsCB</td>
</tr>
<tr>
<td>ovwConfirmCreateObjects</td>
<td>Objects created on map</td>
<td>OVwConfirmCreateObjectsCB</td>
</tr>
<tr>
<td>ovwConfirmCreateSubmaps</td>
<td>Submaps created on map</td>
<td>OVwConfirmCreateSubmapsCB</td>
</tr>
<tr>
<td>ovwConfirmMoveSymbol</td>
<td>Symbol moved</td>
<td>OVwConfirmMoveSymbolCB</td>
</tr>
<tr>
<td>ovwConfirmManageObjects</td>
<td>Objects managed</td>
<td>OVwConfirmManageObjectsCB</td>
</tr>
<tr>
<td>ovwConfirmUnmanageObjects</td>
<td>Objects unmanaged</td>
<td>OVwConfirmManageObjectsCB</td>
</tr>
<tr>
<td>ovwConfirmHideSymbols</td>
<td>Symbols hidden</td>
<td>OVwConfirmHideSymbolsCB</td>
</tr>
<tr>
<td>ovwConfirmUnhideSymbols</td>
<td>Symbols unhidden</td>
<td>OVwConfirmHideSymbolsCB</td>
</tr>
<tr>
<td>ovwConfirmSymbolStatusChange</td>
<td>Symbol status change</td>
<td>OVwConfirmObjectStatusCB</td>
</tr>
<tr>
<td>ovwConfirmObjectStatusChange</td>
<td>Object status change</td>
<td>OVwConfirmObjectStatusCB</td>
</tr>
<tr>
<td>ovwConfirmCompoundStatusChange</td>
<td>Compound object status change</td>
<td>OVwConfirmObjectStatusCB</td>
</tr>
<tr>
<td>ovwConfirmCapabilityChange</td>
<td>Object capability field change</td>
<td>OVwConfirmCapabilityChangeCB</td>
</tr>
<tr>
<td>ovwUserSubmapCreate</td>
<td>Submap creation notification</td>
<td>OVwUserSubmapCreateCB, OVwAckUserSubmapCreate</td>
</tr>
<tr>
<td>ovwSubmapOpen</td>
<td>Submap open</td>
<td>OVwSubmapOpenCB</td>
</tr>
<tr>
<td>ovwSubmapClose</td>
<td>Submap close</td>
<td>OVwSubmapCloseCB</td>
</tr>
</tbody>
</table>

### Libraries
When compiling a program that uses any of the callbacks corresponding to NetView events, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

### Files
OV\ovw.h

### Related Information
- See "OVwAddCallback" on page 382.
- See "OVwApiIntro" on page 395.
OVwFileDescriptor

Purpose

Accesses the NetView 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 program. Among other things, it could be used by an application's own select to determine if there is input from the NetView program session to process.

Return Values

If successful, OVwFileDescriptor returns a non-negative integer. If unsuccessful, it returns −1 (negative one).

Implementation Specifics

OVwFileDescriptor supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwFileDescriptor, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See [netview](#) on page 65.
- See [OVwInit](#) on page 564.
- See [OVwPending](#) on page 589.
- See [OVwProcessEvent](#) on page 590.
- See [OVwApiIntro](#) on page 395.
OVwFindMenuitem

Purpose
Finds a menu item.

Syntax
#include <OV\ovw.h>
#include <OV\ovw_reg.h>

char *OVwFindMenuitem(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 user 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 user interface of the cascades and buttons for the menu item, separated by the current menu path separator. See OVwSetMenuPathSeparator in "OVwGetMenuPathSeparator" on page 548. The default menu path separator is →.

Return Values
If successful, OVwFindMenuitem returns a non-NULL character pointer. If unsuccessful, it returns a character pointer. Because the return value for OVwFindMenuitem is dynamically allocated, free the string when it is no longer needed.

Error Codes
OVwFindMenuitem sets the error code value that OVwError returns. The following list describes the possible errors:

[OVw_CONNECTION_LOST]
The connection to the NetView 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 GUI API has not been initialized with OVwInit.

Examples
You can use the following code fragment to locate the ID of a menu item:

```c
char *id = OVwFindMenuitem("Administer→Telnet (aixterm)...");
if (id == NULL) {
    fprintf(stderr, "error: \n", OVwErrorMsg(OVwError()));
    return -1;
}
```
Implementation Specifics

OVwFindMenuItem supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwFindMenuItem, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information

- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwCreateMenuItem" on page 446.
- See "OVwSetMenuPathSeparator" in OVwGetMenuPathSeparator on page 548.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
OVwGetAppConfigValues

Purpose

Returns application configuration parameters.

Related Functions

OVwSetAppConfigValues

Syntax

```c
#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 on page 395](#) for an overview of the GUI 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, a NULL pointer is returned.
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 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 GUI API has not been initialized with OVwInit.

**Examples**

The following example illustrates how to check the application’s configuration field values:

```c
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 multibyte character code sets.

**Libraries**

When compiling a program that uses OVwGetAppConfigValues or OVwSetAppConfigValues, link to the following libraries:

- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`
- `/usr/ov/lib/libnti.lib`

**Related Information**

- See [“netview” on page 65](#)
- See [“OVwError” on page 520](#)
• See "OVwGetMapInfo" on page 544.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
• See "OVwRegIntro" on page 591.
• See "OVwDbSetFieldValue" on page 508.
OVwGetAppName

Purpose
Returns the name of the running application.

Syntax
```c
#include <OV\ovw.h>
char *OVwGetAppName();
```

Description
OVwGetAppName returns the name of the currently running application as it was registered with the NetView program. This name is used in other NetView 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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwGetAppName supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwGetAppName, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information
- See [netview](#) on page 65
- See [OVwError](#) on page 520
- See [OVwInit](#) on page 564
- See [OVwApiIntro](#) on page 395
OVwGetClientList

Purpose

Returns a list of clients attached to the server.

Syntax

OVwGetClientList(0,0)

Description

Returns a list of IP addresses that identify the clients attached to ovwdb on the server. The `ovobjprint -C` command uses OVwGetClientList to list clients.

Examples

```c
clientAddr = OVwGetClientList(0,0);
printf("ovwdb has the following clients: \r\n");
while(*clientAddr != 0)
{
    //const char *ClientName = OVsnmpHostName(*clientAddr);
    unsigned char addr[4];
    memcpy(addr, clientAddr, sizeof(int));
    printf("\t%u.%u.%u.%u\r\n", (unsigned)addr[0], (unsigned)addr[1], (unsigned)addr[2], (unsigned)addr[3]);
    clientAddr++;
}
```

If you want to print the name (instead of the address) of the client, use the line `char *ClientName = OVsnmpHostName(clientAddr);`
OVwGetConnSymbol

Purpose

Returns a connection symbol.

Syntax

```c
#include <OV\ow.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 on the open map.

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 metaconnection symbol.

OVwFreeSymbolInfo should be used to free the OVwSymbolInfo structure returned by OVwGetConnSymbol. See [OVwGetSymbolInfo](#) on page 558 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, NULL is returned.

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 program was lost.

- **[OVw_CONN_SYMBOL_BOTH_ENDS_NULL]**
  Both connection endpoints have the value ovwNullSymbolId.
[OVw_CONN_SYMBOL_BOTH_ENDS_SAME]
Both connection endpoints are the same.

[OVw_CONN_SYMBOL_END_NOT_FOUND]
One of the connection endpoint 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 endpoints 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 GUI 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:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetConnSymbol, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See “netview” on page 65
- See “OVwCreateSymbol” on page 453
- See “OVwError” on page 520
- See “OVwGetMapInfo” on page 544
- See “OVwGetSymbolInfo” on page 558
- See “OVwInit” on page 564
- See “OVwListSymbols” on page 574
- See “OVwApiIntro” on page 395
OVwGetFirstAction

Purpose

Obtains registered application actions.

Related Functions

OVwGetNextAction

Syntax

```c
#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" on page 552.

OVwGetFirstAction returns a pointer to the name of the first action registered in the current registration context. It should be called before OVwGetNextAction to start 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, 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI 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:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetFirstAction or OVwGetNextAction, link to the following libraries:

- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwCreateAction" on page 436
- See "OVwGetAction in "OVwCreateAction" on page 436
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See "OVwCreateApp" on page 440
- See "OVwGetRegContext" on page 552
- See "OVwCreateAction" on page 436
- See "OVwErrorMsg" on page 521
OVwGetFirstMenuItem

Purpose

Obtains registered menu items.

Related Functions

OVwGetNextMenuItem

Syntax

```c
#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, 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" on page 552.

OVwGetFirstMenuItem should be called before OVwGetNextMenuItem to start 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 the action Get 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 returned.

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 is 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 is 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 is 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 `notepad`, this is equivalent to specifying `!"notepad"` 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, 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI 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 multibyte character code sets.

**Libraries**

When compiling a program that uses `OVwGetFirstMenuItem` or `OVwGetNextMenuItem`, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

**Files**

`OV\ovw.h`

**Related Information**

- See `netview" on page 65`.
- See `OVwError" on page 520`.
- See `OVwInit" on page 564`.
- See `OVwCreateMenuItem" on page 446`.
- See `OVwApiIntro" on page 395`. 
See "OVwRegIntro" on page 591.
See "OVwCreateApp" on page 440.
See "OVwGetRegContext" on page 552.
OVwGetFirstMenuItemFunction

Purpose

Obtains functions bound to a menu item.

Related Functions

OVwGetNextMenuItemFunction

Syntax

```c
#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
the functions bound to registered menu items, in the current registration context. 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 the OVwGetNextMenuItemFunction to start the function traversal.

OVwGetNextMenuItemFunction returns the next function bound to the specified
menu item in the current registration context. This action should be repeated until it
returns a NULL indicating that all application names have been returned.

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 notepad, this would be equivalent to specifying ! notepad for the
menu item in the application registration file.

menuItemld

A pointer to a menu item identifier returned from an OVwFindMenuItem call.

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 program was lost.

[OVw_MENUITEM_NOT_FOUND]
The specified menuItemld 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 GUI 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetFirstMenuItemFunction or
OVwGetNextMenuItemFunction, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Files

OV\ovw.h

Related Information

- See “netview” on page 62
- See “OVwError” on page 520
- See “OVwInit” on page 564
- See “OVwCreateMenuItem” on page 446
- See “OVwApiIntro” on page 395
- See “OVwRegIntro” on page 591
- See “OVwFindMenuItem” on page 525
OVwGetFirstRegContext

Purpose

Gets registered applications.

Related Functions

OVwGetNextRegContext

Syntax

```c
#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 registration contexts, that is, the name of each application that is registered with the NetView program.

OVwGetFirstRegContext returns the name of the first application in the NetView program’s list of registered applications. It should be called before OVwGetNextRegContext to start the name traversal.

OVwGetNextRegContext returns the next registration context name in the NetView 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, 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 program was lost.

- `[OVw_OUT_OF_MEMORY]`
  A memory allocation failure occurred.

- `[OVw_OVW_NOT_INITIALIZED]`
  The GUI API has not been initialized with OVwInit.
All application names have been obtained.

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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetFirstRegContext or OVwGetNextRegContext, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "netview" on page 63
- See "OVwError" on page 520
- See "OVwGetRegContext" on page 552
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See OVwGetApp in "OVwCreateApp" on page 440
- See "OVwErrorMsg" on page 521
- See "OVwCreateApp" on page 440
OVwGetMapInfo

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 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" on page 585.

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 program was lost.
[OVw_MAP_NOT_OPEN]
There is no open map.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwGetMapInfo and its related functions support single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwGetMapInfo or one of its related functions, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information
- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwMapOpenCB" on page 585.
- See "OVwApiIntro" on page 395.
OVwGetMenuItemPath

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 user 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, 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 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 GUI API has not been initialized with OVwInit.
Implementation Specifics

OVwGetMenuItemPath and OVwGetMenuItemMenu support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwGetMenuItemPath or OVwGetMenuItemMenu, link to the following libraries:

- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwFindMenuItem" on page 525
- See "OVwCreateMenuItem" on page 446
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See "OVwGetMenuPathSeparator" on page 548
OVwGetMenuPathSeparator

Purpose

Obtains the menu path separator string.

Related Functions

OVwSetMenuPathSeparator

Syntax

#include <OV\ovw.h>
#include <OV\ovw_reg.h>

dchar */OVwGetMenuPathSeparator();

int OVwSetMenuPathSeparator(char */separator);

Description

OVwGetMenuPathSeparator returns the current character string used to separate
menu labels in a menu path string representing the path of a menu item on the
menu bar.

OVwSetMenuPathSeparator sets the menu path separator string to the specified
value. The default menu separator string is +.

These routines affect how parameters to other registration routines behave by
changing the way menu path names are expressed. See OVwFindMenuItem on
page 523 and OVwGetMenuItemMenu in OVwGetMenuItemPath on page 546.
These changes affect the way menu path names are expressed for the NetView
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 non-alphabetic, 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, free the string when it is no longer needed.

If successful, OVwSetMenuPathSeparator returns 0 (zero); if unsuccessful, −1
(negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.
The GUI API has not been initialized with OVwInit.

Examples

```c
OVwSetMenuPathSeparator("==");
char *id = OVwFindMenuItem("Tools.==Telnet");
if (id != NULL) {
    fprintf(stderr, "error: %s\n", OVwErrorMsg(OVwError()));
    return -1;
}
```

Implementation Specifics

OVwGetMenuPathSeparator and OVwSetMenuPathSeparator support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwGetMenuPathSeparator or OVwSetMenuPathSeparator, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
- See "OVwFindMenuItem" on page 525.
- See "OVwGetMenuItemPath" on page 546.
- See "OVwErrorMsg" on page 521.
**OVwGetObjectInfo**

**Purpose**

Returns map-specific object information.

**Related Functions**

OVwFreeObjectInfo

**Syntax**

```c
#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 be freed.

- **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 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 GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwGetObjectInfo supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwGetObjectInfo or OVwFreeObjectInfo, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib

Related Information
• See "netview" on page 65
• See "OVwDbCreateObject" on page 471
• See "OVwDbSelectionNameToObjectIId" on page 504
• See "OVwError" on page 520
• See "OVwGetMapInfo" on page 544
• See "OVwInit" on page 564
• See "OVwApiIntro" on page 395
OVwGetRegContext

Purpose

Retrieves the application registration context.

Related Functions

OVwSetRegContext

Syntax

```c
#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 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, free the string when it is no longer needed.

If successful, OVwSetRegContext returns 0 (zero); if unsuccessful, −1 (negative one) is returned.

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.
The connection to the NetView program was lost.

A memory allocation failure occurred.

The GUI API has not been initialized with OVwInit.

Examples

The following example shows how the routines can be used to retrieve application registration information for NetView applications. The example iterates all applications and prints some registration information for each application:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetRegContext or OVwSetRegContext, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "netview" on page 63
- See "OVwError" on page 520
- See OVwGetApp in "OVwCreateApp" on page 440
- See "OVwGetFirstRegContext" on page 542
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See "OVwErrMsg" on page 524
- See "OVwGetAppName" on page 530
- See "OVwGetFirstRegContext" on page 542
- See "OVwCreateApp" on page 440
OVwGetSelections

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 specified 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 oovwMapOpen 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, a NULL pointer is returned.

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 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 GUI 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

The following code fragment illustrates highlighting the current list of selected objects:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwGetSelections, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
- See OVwDbFreeObjectIdList in "OVwDbListObjectsByFieldValue" on page 499
- See "OVwGetMapInfo" on page 544
- See "OVwHighlightObject" on page 562
OVwGetSubmapInfo

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.
Use OVwFreeSubmapInfo 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 be freed

submapId
Specifies the submap ID of the submap

Return Values
If successful, OVwGetSubmapInfo returns a pointer to an OVwSubmapInfo
structure; if unsuccessful, NULL is returned.

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 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 GUI 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 multibyte character code sets.

Libraries
When compiling a program that uses OVwGetSubmapInfo or OVwFreeSubmapInfo, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information
- See “netview” on page 65.
- See “OVwCreateSubmap” on page 449.
- See “OVwError” on page 520.
- See “OVwGetMapInfo” on page 544.
- See “OVwInit” on page 564.
- See “OVwApiIntro” on page 395.
OVwGetSymbolInfo

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, NULL is returned.

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 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 GUI API has not been initialized with OVwInit.
The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwGetSymbolInfo and OVwFreeSymbolInfo support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwGetSymbolInfo or OVwFreeSymbolInfo, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See “netview” on page 65
- See “OVwCreateSymbol” on page 453
- See “OVwError” on page 520
- See “OVwGetMapInfo” on page 544
- See “OVwInit” on page 564
- See “OVwApiIntro” on page 395
**OVwGetSymbolsByObject**

**Purpose**

Returns symbols for an object.

**Syntax**

```c
#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 on page 574](#), 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 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 GUI API has not been initialized with OVwInit.

Libraries

When compiling a program that uses OVwGetSymbolsByObject, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See netview on page 65
- See OVwError on page 520
- See OVwGetMapInfo on page 544
- See OVwInit on page 564
- See OVwListSymbols on page 574
- See OVwApiIntro on page 395
- See OVwFreeSymbolList in OVwListSymbols on page 574
OVwHighlightObject

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 user 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 user 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 user 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, −1 (negative one) is returned.

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 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 GUI API has not been initialized with OVwInit.

Examples

The following example shows how an application can highlight the current list of selected objects:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwHighlightObject or OVwHighlightObjects, link to the following libraries:

- `\us\ov\lib\libovw.lib`
- `\us\ov\lib\libov.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See OVwDbFreeObjectIdList in "OVwDbListObjectsByFieldValue" on page 498
- See "OVwGetMapInfo" on page 544
- See "OVwGetSelections" on page 552
OVwInit

Purpose

Initializes an application’s connection to the NetView program.

Syntax

```c
#include <OV\ovw.h>
int OVwInit();
```

Description

OVwInit initializes internal API data structures and the communications channel between an NetView application and the NetView program. It must be called before any other GUI API call.

Return Values

If successful, OVwInit returns 0 (zero); if unsuccessful, −1 (negative one) is returned.

Error Codes

OVwInit sets the error code value that OVwError returns. The following list describes the possible errors:

- **[OVw_ALREADY_INITIALIZED]**
  The GUI API has been initialized with a prior call to OVwInit.

- **[OVw_CONNECT_ERROR]**
  A failure occurred when attempting to connect to the NetView program.

- **[OVw_CONNECTION_LOST]**
  The connection to the NetView program was lost.

- **[OVw_OVW_NOT_RUNNING]**
  The application was not invoked from the NetView program.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

Examples

The following code fragment shows how to initialize the GUI API prior to other OVw calls:

```c
if (OVwInit() < 0) {
    fprintf(stderr, "OVwInit: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
}
OVwAddCallback(ovwEndSession, NULL, (OVwCallbackProc)endCB, NULL);
```

Libraries

When compiling a program that uses OVwInit, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
Related Information

- See "netview" on page 63.
- See "OVwDone" on page 517.
- See "OVwError" on page 520.
- See "OVwApiIntro" on page 395.
- See "OVwErrorMsg" on page 521.
- See "OVwAddCallback" on page 382.
**OVwInitApiEvents**

**Purpose**
Registers a messaging application to receive NetView events.

**Syntax**
```
#include <OV\ovw.h>

void OVwInitApiEvents();
```

**Description**
OVwInitApiEvents allows a messaging application to receive NetView events. A Windows or other messaging application must make this call in order to receive NetView callbacks.

**Error Codes**
OVwInitApiEvents sets the error code value the OVwError returns. The following list describes the possible errors:

- **[OVw_ALREADY_INITIALIZED]**
  The GUI API has been initialized with a prior call to OVwInit.

- **[OVw_CONNECTION_LOST]**
  The connection to the NetView program was lost.

- **[OVw_NOT_INITIALIZED]**
  The GUI API was not initialized with OVwInit.

**Libraries**
When compiling a program that uses OVwApiEvents, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

**Related Information**
See "OVwInit" on page 564.
OVwIsIdNull

**Purpose**
Tests and compares GUI API IDs.

**Related Functions**
OVwIsIdEqual

**Syntax**

```c
#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 GUI 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 \textit{id} has a null value; otherwise, it returns FALSE.

OVwIsIdEqual is a macro that returns TRUE if \textit{id1} and \textit{id2} are equal; otherwise, it returns FALSE. Both IDs should be of the same type.

**Parameters**

- \textit{id} specifies an object ID, field ID, submap ID, or symbol ID.
- \textit{id1} specifies an object ID, field ID, submap ID, or symbol ID.
- \textit{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 GUI API is changed.

**Libraries**

When compiling a program that uses OVwIsIdNull or OVwIsIdEqual, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

**Files**

- `OV\ovw.h`
- `OV\ovw_types.h`
Related Information

See "OVwApiIntro" on page 395.
OVwListObjectsOnMap

Purpose
Lists objects on a map.

Related Functions
OVwFreeObjectList

Syntax
#include <OV\ovw.h>

OVwObjectList *OVwListObjectsOnMap(OVwMapInfo *map,
    OVwFieldBindList *fieldValues);

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 the 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 program was lost.

[OVw_DB_CONNECTION_LOST]
    The connection to the ovwdb object database was lost.
[OVw.getFieldNotFound]  
A field ID in the fieldValues argument does not indicate a field in the database.

[OVw.getFieldTypeMismatch]  
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.getMapNotOpen]  
The argument map does not specify an open map.

[OVw.getOutOfMemory]  
A memory allocation failure occurred.

[OVw.getOVwNotInitialized]  
The GUI API has not been initialized with OVwInit.

Implementation Specifics

OVwListObjectsOnMap and OVwFreeObjectList support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwListObjectsOnMap or OVwFreeObjectList, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See “netview” on page 65.
- See “ovwdb” on page 464.
- See “OVw.Error” on page 520.
- See “OVw.GetMapInfo” on page 544.
- See “OVw.Init” on page 564.
- See “OVw.ApiIntro” on page 395.
OVwListSubmaps

Purpose

Lists submaps on a map.

Related Functions

OVwFreeSubmapList

Syntax

```c
#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. A NULL `appName` 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 be freed.

- `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 program was lost.

[OVw_DB_CONNECTION_LOST]
   The connection to the ovwdb object database was lost.

[OVw_FIELD_NOT_FOUND]
   A field ID in the parentFieldValues argument does not indicate a field in the database.

[OVw_FIELD_TYPE_MISMATCH]
   The field data type provided in an OVwFieldBinding structure in parentfieldValues 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 GUI API has not been initialized with OVwInit.

Examples

The following example shows how to list all submaps on the open map:

```c
OVwListSubmaps(map, NULL, ovwAnySubmapType, NULL);
```

The following example shows how to list all submaps created by the calling application:

```c
char *appname = OVwGetAppName();
OVwListSubmaps(map, appname, ovwAnySubmapType, NULL);
```

The following example shows how to list all submaps of type 2 created by the IP Map application:

```c
OVwListSubmaps(map, "IP Map", 2, NULL);
```

The following example shows how to print the names of all submaps on the open map:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwListSubmaps or OVwFreeSubmapList, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See "netview" on page 65.
- See "ovwdb" on page 464.
- See "OVwError" on page 520.
- See "OVwGetAppName" on page 530.
- See "OVwGetMapInfo" on page 544.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
**OVwListSymbols**

**Purpose**

Lists symbols on a submap.

**Related Functions**

OVwFreeSymbolList

**Syntax**

```c
#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, NULL is returned. The number of items in the symbol list may 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 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 GUI 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
The following example shows how to list all symbols on the specified submap:
OVwListSymbols(map, submap_id, ovwAllPlanes, NULL);

The following example shows how to list all symbols of the calling application:
char *appname = OVwGetAppName();
OVwListSymbols(map, submap_id, ovwAppPlane, appname);

Implementation Specifics
OVwListSymbols and OVwFreeSymbolList support single-byte and multibyte character code sets.
Libraries

When compiling a program that uses OVwListSymbols or OVwFreeSymbolList, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information

- See “netview” on page 65.
- See “OVwError” on page 520.
- Refer to Tivoli NetView for Windows “OVwGetAppName” on page 530.
- Refer to Tivoli NetView for Windows “OVwGetMapInfo” on page 544.
- See “OVwInit” on page 564.
- See “OVwSetSymbolApp” on page 628.
- See “OVwApiIntro” on page 395.
- See “OVwRegIntro” on page 591.
 OVwListSymbolTypes

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](on page 453) 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, NULL is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]

A memory allocation failure occurred.
[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwListSymbolTypes and its related functions support single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwListSymbolTypes or its related functions, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib

Related Information
• See "netview" on page 65
• See "OVwCreateSymbol" on page 453
• See "OVwError" on page 520
• See "OVwInit" on page 564
• See "OVwApiIntro" on page 395
• See OVwDbFreeFieldBindList in "OVwDbGetFieldValues" on page 486
OVwLockRegUpdates

Purpose

Acquires permission to modify registration information.

Related Functions

OVwUnlockRegUpdates

Syntax

```c
#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 registration information. Such calls include the OVwAddMenuItem and OVwAddMenuItemFunction. Only one NetView 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, −1 is returned.

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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI 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, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib

Related Information
• See "netview" on page 65.
• See "OVwAddMenuItemFunction" on page 390.
• See "OVwError" on page 520.
• See "OVwDone" on page 517.
• See "OVwInit" on page 564.
• See "OVwApiIntro" on page 395.
• See "OVwRegIntro" on page 591.
OVwMainLoop

Purpose

Defines NetView graphical user interface MainLoop, which continuously processes NetView events.

Syntax

```c
#include <OV\ovw.h>
void OVwMainLoop();
```

Description

OVwMainLoop is a macro that defines a while loop that continuously processes NetView events and application-registered input events. Applications must either call OVwMainLoop or OVwInitApiEvents for application callbacks, registered with OVwAddCallback, to be functional.

OVwMainLoop returns only if the connection to the NetView 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 application uses OVwMainLoop:

```c
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, "OVwInit: %s\n", OVwErrorMsg(OVwError()));
    exit(1);
  }
  OVwAddCallback(ovwEndSession, NULL, (OVwCallbackProc)EndSessionProc, NULL);
  OVwMainLoop();
}
```

Libraries

When compiling a program that uses OVwMainLoop, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
Related Information

- See "netview" on page 66.
- See "OVwAddCallback" on page 382.
- See "OVwAddInput" on page 386.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwErrorMsg" on page 521.
- See "OVwDone" on page 517.
**OVwMapCloseCB**

**Purpose**

Functions as a callback for a map close event.

**Syntax**

```c
#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 user 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 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.

**Note:** Because application acknowledgement is required to close a map, response time might be slow if applications are busy. To avoid this, it is 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. When a map close event is imminent through OVwPeekOVwEvent, the application can determine the map close time.

There is a map close timeout, so that if all applications do not close within two minutes, the map is closed anyway.
**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:

```c
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 multibyte character code sets.

**Libraries**

When compiling a program that uses `OVwMapCloseCB`, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

**Related Information**

- See "netview" on page 65
- See "OVwAddCallback" on page 383
- See "OVwMapOpenCB" on page 585
- See "OVwPeekOVwEvent" on page 587
- See "OVwApiIntro" on page 395
- See "OVwSetStatusOnObject" on page 623
- See "OVwBeginMapSync" on page 410
OVwMapOpenCB

Purpose

Functions as a callback for a map open event.

Syntax

```c
#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" on page 623 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" on page 544) 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:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwMapOpenCB, link to the following
libraries:
   • `\usr\ov\lib\libovw.lib`
   • `\usr\ov\lib\libov.lib`

Related Information

   • See `OVwAddCallback` on page 383.
   • See `OVwBeginMapSync` on page 410.
   • See `OVwGetMapInfo` on page 544.
   • See `OVwSetStatusOnObject` on page 623.
   • See `OVwMapCloseCB` on page 583.
   • See `OVwApiIntro` on page 395.
   • See `OVwRegIntro` on page 591.
OVwPeekOVwEvent

Purpose

Checks for specific NetView 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 event is awaiting processing. This is particularly helpful in applications that must handle the ovwMapClose event promptly. By checking for the ovwMapClose event at certain intervals during lengthy processing, you can quickly stop processing so that the map close event can be received and acknowledged promptly. See "OVwAckMapClose" on page 373 and "OVwMapCloseCB" on page 583 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 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 events, OVwPeekOVwEvent returns TRUE. If there are no pending input events, OVwPeekOVwEvent returns FALSE.

If there are pending input events, OVwPeekInputEvent returns TRUE. If there are no pending NetView 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 program was lost.

[OVw_OUT_OF_MEMORY]

A memory allocation failure occurred.
The GUI 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:

```c
OVwBeginMapSync(map);
synching = TRUE;
mapClosing = FALSE;

while (synching) {
    if (OVwPeekOVwEvent(ovwMapClose) == TRUE) {
        synching = FALSE;
        mapClosing = TRUE;
    } else {
        /* Process a few map synchronization steps */
    }
}

OVwEndMapSync(map);
```

Libraries

When compiling a program that uses OVwPeekOVwEvent or OVwPeekInputEvent, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Files

OV\ovw.h

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwPending" on page 589
- See "OVwApiIntro" on page 395
- See "OVwAckMapClose" on page 373
- See "OVwMapCloseCB" on page 583
- See "OVwAddInput" on page 386
- See "OVwBeginMapSync" on page 410
OVwPending

Purpose
Tests for pending NetView or application-registered events.

Syntax
#include <OV\ovw.h>

OVwBoolean OVwPending();

Description
OVwPending returns a boolean value indicating whether there is an NetView 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 process was lost.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Libraries
When compiling a program that uses OVwPending, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib

Related Information
- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwInit" on page 564
- See "OVwProcessEvent" on page 590
- See "OVwApiIntro" on page 395
OVwProcessEvent

Purpose

Processes a pending NetView or application-defined event.

Syntax

#include <OV\ovw.h>

int OVwProcessEvent();

Description

OVwProcessEvent processes pending NetView 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, -1 is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]  
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]  
The GUI 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, link to the following libraries:

• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib

Related Information

• See "netview" on page 65
• See "OVwError" on page 520
• See "OVwInit" on page 564
• See "OVwPending" on page 589
• See "OVwApiIntro" on page 395
• See "OVwAddCallback" on page 383
• See "OVwAddActionCallback" on page 377
• See "OVwAddInput" on page 386
• See "OVwMainLoop" on page 581
OVwRegIntro

Purpose

Provides an overview of NetView graphical user interface registration files.

Description

NetView graphical user interface registration files are a mechanism for:

- Integrating applications into the NetView graphical user interface
- Defining symbol types for the NetView graphical user interface map
- Creating fields in the NetView graphical user interface Object Database (see "ovwrdb" on page 484)
- Enabling managers to access your agent or application through the NetView program

When the NetView 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 program’s -verify option to ensure the accuracy of the registration files before starting a NetView session if registration files have been modified. See "netview" on page 65.

Application Integration

Applications that are started from the NetView program and those that use the GUI API must be registered through an application registration file. The file provides the necessary information for the NetView program to invoke and manage the application processes.

Application registration files define how an application integrates with NetView graphical user interface. This information includes details, such as:

- The definition of specific actions the application can perform
- A command to start the application
- Flags telling the NetView program how the application command should be started and managed
- Graphical user 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

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 GUI API or the graphical user interface.

Some applications manage semantics, or semantic planes, of graphical user 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 GUI API are available in the Tivoli NetView for Windows Programmer’s Guide.

Application registration files are located in the \usr\ov\reg\advanced\c, \usr\ov\reg\beginner\c, \usr\ov\registration\c directories. For non-English versions of
NetView the "c" subdirectory would be replaced by the language-specific subdirectory name. The NetView 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 graphical user interface menu registration is processed first, followed by all other applications in application name order. That is, the application Alpha's menu items are registered before the Omega's menu items.

**Application Block**

The Application block contains information about how an application is integrated with the NetView program. Only one application can be registered in a single application 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**

```plaintext
Application "Sample App" {
  Command "/usr/local/bin/checknode -test";
  
}
```

**Second sample application registration**

```plaintext
Application "Sample App" {
  Command "/usr/bin/foo -x -y -z ${OVwSelections}";
  
}
```

The NetView 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 application executes the command string. For the command invocation to succeed, the first element of the command string must be the path of an executable file. The NetView program sets the following environment variables.

- **OVwSelections**
  This is a list, separated by blanks, of the object names of the objects in the selection list at the time the application is invoked.

- **OVwNumSelections**
  This is set to the number of objects in the selection list at the time the application is invoked.

- **OVwSelectionn (n=1, 2, ..., 10)**
  Each of these environment variables contains the object 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 object name of the tenth object selected. The ordering of the selected objects for these variables is the order in which they were selected on the map.

**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 program how to manage the application process or processes. Flags are specified with the command in the application block, as shown in the following example:

```plaintext
Application "Sample App" {
  Command -Shared -Initial -Restart "/usr/bin/foo -x -y -z ${OVwSelections}";
}
```

Valid process flags are:

- **-Initial**
  This flag signals the NetView program to start the application when the NetView program starts. The NetView 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 signals the NetView 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 GUI API when actions are requested by menu items or executable symbols. Applications that do not use the GUI 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 GUI API to intercept further action requests from menu items or executable symbols.

- **-Restart**
  This flag signals the NetView 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 user interface maps and should always be present for the duration of an NetView session.
The Description block provides a brief description of the application used in the Help..Applications dialog box, which displays all the applications installed and registered with the NetView program.

The description block contains a comma-separated list of strings. Within the application index, each string is displayed on a separate line.

**Application Copyright**

The Copyright block provides the copyright string used in "Help..Applications" dialog box. Like the Application Description, this block contains a comma-separated list of strings. Within the application index, each string is displayed on a separate line.

**Application Version**

The version statement defines a string which is the application version information. This version string is displayed in the Help..On Version and Help..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 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\c.

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 object name. The environment variable $OVwSelections contains, by default, the object 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 object 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:

```c
Application "Notepad" {
  Command "/winnt/system32/notepad ${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, its object name is used.
Application Menu Integration

The MenuBar and Menu sections provide a specification of how the application interfaces with the graphical user interface menu bar. They 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 (see "Menu Registration").

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 {

}
```

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 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. Menu bar cascades are listed in the menu bar according to precedence and, within items of the same precedence, the order in which they were registered (collated order of application name). As shown in the following example, the NetView 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:

```plaintext
MenuBar <100> "File" _F {
    
}
```

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:

```plaintext
Application "Mail Manager"
{
    MenuBar "Monitor"
    {
        "Mail" f.menu "MailMenu";
    }
    ObjectMenu
    {
        "MenuItem" f.action "MenuItem";
        "MenuItem" f.action "MenuItem";
    }
}
```

In the preceding 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 user 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 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 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 user 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. The [menu example on page 596](#) shows “f.menu” declaring the “MailMenu” cascade.

**f. “built-in-function-name”**

Built-in functions are internal to the NetView program. They are functions that implement internal NetView callbacks, providing functionality for menu items, such as Help..On Version, File..New Map..., and so on. These are provided so that the NetView 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 registration file contains several examples of internal functions, such as “f.star_center”.

**Note:** These internal functions are designed for the NetView 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 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 GUI 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.

In the [Menu Registration on page 595](#), 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:
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 logical 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 the grammar beginning on page 604 for the detailed syntax of the SelectionRule expression.

**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 need to 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.

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/bar -x";
  }
}
```

Each command is considered separately because the process flags differ. The “Foo” action command is shared across actions and is started with the NetView 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 GUI API.

**Application Field Enrollment**

The field enrollment section is needed only by applications that will manage the semantics of graphical user 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.

**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" on page 652.
- **Describe** The dialog box presented when describing an object on the map. See "OVwVerifyDescribeChange" on page 668.
- **Connect** The dialog box presented when connecting two objects on the map. See "OVwVerifyConnect" on page 661.
- **Configuration** The dialog box presented per map for configuring the application. This dialog box is special in that it is not associated with a particular object. See "OVwVerifyAppConfigChange" on page 658.

**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.

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 the parameters supplied 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 be displayed in the dialog box and how those fields should be presented.

See page 513 for an example field registration file.

A Symbol Class registration file provides a way to define a new class of symbols.

See page 512 for an example symbol class and symbol type registration file.

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 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 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 user 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 user 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 displays in the Locate by Properties dialog box.

- **General**
  Appears in the general properties 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 | ";"
```
Chapter 2. Reference Pages
Integer ::= one or more decimal digits | "-" one or more decimal digits

Menu ::= "Menu" MenuID MenuBlock
MenuID ::= StringOrIdentifier
ObjectMenu ::= "ObjectMenu" MenuBlock
Tool ::= "Tool" 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 ::= SelectionSpecifier | SelectionSpecifierList "," 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
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"
IntegerFieldDisplayPolicy ::= "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 "}"

610  Programmer's Reference
Example

Application "MIB Browser"
{
  Description {
    "Browser for SNMP MIB Data"
  }

  Version "V2.1 (Rev 35)";

  Copyright {
    "Licensed Program Product: POLYCENTER Manager on NetView for Windows",
    "(C) COPYRIGHT International Business Machines Corp. 1992,1993",
    "(C) COPYRIGHT Hewlett-Packard Co. 1992",
    "All Rights Reserved",
    ""
  }

  /*
   * MIB Browser is not -Shared so that a separate copy can be
   * invoked to compare values from different nodes.
   */
  Command "/usr/ov/bin/browser.exe -node "${OVwSelections}";

  MenuBar <20> "Tools"
  {
    <100> "MIB" _M f.menu "MIB";
  }
}
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 graphical user interface registration files, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libovw.lib

Implementation Specifics

The environment variable, LANG, determines the language in which messages are displayed. If LANG is not specified or is set to the empty string, the default C is used. If any internationalization variable contains a setting that is not valid, the OVwRegIntro command functions as though all internationalization variables are set to C.

The OVwRegIntro command supports single-byte and multibyte character code sets.

Related Information

- See "netview" on page 63
- See "ovwdb" on page 464
• Refer to the *Programmer’s Guide*
• Refer to the *User’s Guide*
• See "OVwVerifyAdd" on page 652.
• See "OVwVerifyDescribeChange" on page 668.
• See "OVwVerifyConnect" on page 661.
• See "OVwVerify AppConfigChange" on page 658.
**OVwRenameRegContext**

**Purpose**
Changes the name of a registration context.

**Syntax**

```c
#include <OV\ovw.h>
#include <OV\ovw_reg.h>

int OVwRenameRegContext(char *fromAppName, char *toAppName);
```

**Description**

OVwRenameRegContext changes the name of a registered NetView 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 program was lost.

- **[OVw_OUT_OF_MEMORY]**
  A memory allocation failure occurred.

- **[OVw_OVW_NOT_INITIALIZED]**
  The GUI 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwRenameRegContext, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwGetAppConfigValues" on page 527
- See "OVwGetFirstRegContext" on page 543
- See "OVwInit" on page 564
- See "OVwApIntro" on page 395
- See "OVwRegIntro" on page 591
- See "OVwLockRegUpdates" on page 576
- See "OVwSaveRegUpdates" on page 617
OVwSaveRegUpdates

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 program to save changes that have been made to registration information through the GUI API. The changes affect the current NetView session and can optionally be saved in the appropriate application registration files. GUI API calls that alter application registration information include OVwAddMenuItem and OVwAddMenuItemFunction.

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 user interface session will retain the modifications.

Return Values
If successful, OVwSaveRegUpdates and OVwUndoRegUpdates return 0 (zero). If unsuccessful, −1 (negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

[OVw_PERMISSION_DENIED]
The application either has not called OVwLockRegUpdates prior to this function call.
Libraries

When compiling a program that uses OVwSaveRegUpdates or OVwUndoRegUpdates, link to the following libraries:

- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`

Related Information

- See “netview” on page 65
- See “OVwError” on page 520
- See “OVwInit” on page 564
- See “OVwLockRegUpdates” on page 579
- See “OVwApiIntro” on page 395
- See “OVwRegIntro” on page 591
OVwSelectListChangeCB

Purpose

Functions as a callback for a selection list change event.

Syntax

```c
#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, namely ovwSelectListChange. This is useful if one callback handles multiple events.

- **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:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwSelectListChangeCB, link to the following libraries:

- \usr\ov\lib\libovw.lib
• \\usr\ov\lib\libov.lib

Related Information

• See “netview” on page 65.
• See “OVwAddCallback” on page 382.
• See “OVwApiIntro” on page 395.
• See “OVwGetSelections” on page 554.
• See “OVwHighlightObject” on page 562.
**OVwSetBackgroundGraphic**

**Purpose**
Sets the background graphic for a submap.

**Related Functions**
OVwClearBackgroundGraphic

**Syntax**

```c
#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 must be in the Windows DIB (device independent bitmap) format with the file extension "bmp:"
  
  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, -1 (negative one) is returned.

**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 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 GUI 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 multibyte character code sets.

Libraries
When compiling a program that uses OvwSetBackgroundGraphic or OVwClearBackgroundGraphic, link to the following libraries:

• /usr/ov/lib/libovw.lib
• /usr/ov/lib/libov.lib

Related Information
  • See “netview” on page 65.
  • See “OVwError” on page 520.
  • See “OVwGetMapInfo” on page 544.
  • See “OVwInit” on page 564.
  • See “OVwApiIntro” on page 395.
  • See “OVwGetMapInfo” on page 544.
  • CompuServe Graphics Interchange Format(sm) Version 89a.
**OVwSetStatusOnObject**

**Purpose**

Sets status of objects and symbols.

**Related Functions**

- OVwSetStatusOnObjects
- OVwSetStatusOnSymbol
- OVwSetStatusOnSymbols

**Syntax**

```c
#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 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 specified `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 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.

dsymbolId
Specifies the ID of the symbol.

dsymbolList
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, −1 (negative one) is returned.

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 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 objectld does not exist on the open map.

[OVw_OBJECT_UNMANAGED]
The object indicated by objectld 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 GUI 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 symbolld does not exist on the open map.

[OVw_SYMBOL_STATUS_SOURCE_MISMATCH]
The symbol specified by symbolld does not have status source ovwSymbolStatusSource.

[OVw_SUBMAP_PERMISSION_DENIED]
The symbol specified by symbolld 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 multibyte character code sets.

**Libraries**

When compiling a program that uses OVwSetStatusOnObject or one of its related functions, link to the following libraries:
- `/usr/ov/lib/libovw.lib`
- `/usr/ov/lib/libov.lib`
- `/usr/ov/lib/libntl.lib`

**Files**

OV\ovw_types.h

**Related Information**

- See “netview” on page 65
- See “OVwError” on page 520
- See “OVwGetMapInfo” on page 544
- See “OVwInit” on page 564
- See “OVwSetSymbolStatusSource” on page 640
- See “OVwApiIntro” on page 395
**OVwSetSubmapName**

**Purpose**
Sets name of a submap.

**Syntax**

```c
#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, −1 (negative one) is returned.

**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 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 GUI API has not been initialized with OVwInit.
The submap specified by submapId does not exist on the open map.

The submap is an exclusive submap created by another application.

Implementation Specifics
OVwSetSubmapName supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwSetSubmapName, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

Related Information
- See "netview" on page 63.
- See "OVwCreateSubmap" on page 443.
- See "OVwError" on page 520.
- See "OVwGetMapInfo" on page 544.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
OVwSetSymbolApp

Purpose

Sets application interest in a symbol.

Related Functions

- OVwClearSymbolApp

Syntax

```c
#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 program was lost.
The argument map does not specify an open map.

The map is open with read-only permission.

A memory allocation failure occurred.

The GUI API has not been initialized with OVwInit.

The symbol exists on the application plane of an exclusive submap created by another application.

The symbol specified by symbolId does not exist on the open map.

Implementation Specifics

OVwSetSymbolApp and OVwClearSymbolApp support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwSetSymbolApp or OVwClearSymbolApp, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\linint1.lib`

Related Information

- See "netview" on page 65.
- See "OVwError" on page 520.
- See "OVwGetMapInfo" on page 544.
- See "OVwInit" on page 564.
- See "OVwListSymbols" on page 574.
- See "OVwApiIntro" on page 395.
OVwSetSymbolBehavior

Purpose

Sets the behavior of a symbol.

Syntax

```c
#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, is not registered. See [OVwRegIntro on page 591](#) 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 propagated 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 the 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 the 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, -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 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 GUI 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 multibyte character code sets.

Files
OVW.ovw.h

Libraries
When compiling a program that uses OVwSetSymbolBehavior, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libnt1.lib

Related Information
• See "netview" on page 65
• See "OVwAddActionCallback" on page 377
• See "OVwError" on page 520
• See "OVwGetAppName" on page 530
• See "OVwGetMapInfo" on page 544
• See "OVwInit" on page 564
• See "OVwApiIntro" on page 395
• See "OVwRegIntro" on page 591
OVwSetSymbolLabel

Purpose

Sets the label of a symbol.

Syntax

```c
#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, -1 (negative one) is returned.

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 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 GUI 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 multibyte character code sets.

Libraries
When compiling a program that uses OVwSetSymbolLabel, link to the following libraries:
• \usr\ov\lib\libovw.lib
• \usr\ov\lib\libov.lib
• \usr\ov\lib\libntl.lib

Related Information
• See "netview" on page 65
• See "OVwError" on page 520
• See "OVwGetMapInfo" on page 544
• See "OVwInit" on page 564
• See "OVwApiIntro" on page 395.
OVwSetSymbolPosition

Purpose

Sets the 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” on page 453 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

Chapter 2. Reference Pages 635
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 on page 621." 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 display at a position 25% of the way from the upper lefthand 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 are displayed on the submap. For example, two symbols placed at positions (25, 25) and (75, 75) on a (100 x 100) grid will display 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 seem smaller because of the greater distance between them.

If the symbol is moved on a submap with a nonsequenced 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 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, create the submap 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 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 on page 431" 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, −1 (negative one) is returned.

**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 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 symbolld does not exist on the same submap as the symbol symbolld.

[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 GUI 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 symbolld is displayed.

[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 symbolld does not exist on the open map.

[OVw_SYMBOL_WRONG_VARIETY]
The symbol specified by symbolld does not have variety ovwIconSymbol. Position can only be set for icon symbols.

Implementation Specifics
OVwSetSymbolPosition supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwSetSymbolPosition, link to the following libraries:
  • \usr\ov\lib\libovw.lib
  • \usr\ov\lib\libov.lib
  • \usr\ov\lib\libntl.lib

Related Information
  • See "netview" on page 65.
  • See "OVwConfirmMoveSymbolCB" on page 431.
  • See "OVwCreateSymbol" on page 453.
  • See "OVwError" on page 520.
  • See "OVwGetMapInfo" on page 544.
  • See "OVwInit" on page 564.
  • See "OVwSetBackgroundGraphic" on page 621.
  • See "OVwApiIntro" on page 395.
  • See "OVwCreateSubmap" on page 449.
OVwSetSymbolStatusSource

Purpose

Sets the status source of a symbol.

Syntax

```c
#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 a status source value of 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 a status source value of 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 a status source value of ovwSymbolStatusSource, status can only be set for the symbol using OVwSetStatusOnSymbol or OVwSetStatusOnSymbols, described in "OVwSetStatusOnObject" on page 623. 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 are displayed 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, -1 (negative one) is returned.

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 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 GUI 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 multibyte character code sets.

Files
OV\ovw.h

Libraries
When compiling a program that uses OVwSetSymbolStatusSource, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information
• See "netview" on page 65
• See "OVwError" on page 520
• See "OVwGetMapInfo" on page 544
• See "OVwInit" on page 564
• See "OVwSetStatusOnObject" on page 623.
• See "OVwApiIntro" on page 395.
OVwSetSymbolType

Purpose

Sets the 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 on page 591] 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. This flag 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, −1 (negative one) is returned.

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 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 GUI 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 multibyte character code sets.

Libraries
When compiling a program that uses OVwSetSymbolType, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`
Files

- OV\ovw.h
- OV\sym_types.h

Related Information

- See "netview" on page 65
- See "OVwError" on page 520
- See "OVwGetMapInfo" on page 544
- See "OVwInit" on page 564
- See "OVwApiIntro" on page 395
- See "OVwRegIntro" on page 591
OVwShowHelp

Purpose

Requests presentation of help information.

Syntax

```c
#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 help system display the specified help for the application. The type of request and request string are relayed by the NetView program to the designated NetView 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, which is assumed to be helpRequest. If the application has not registered a help directory, the path is assumed to be relative to the NetView help directory, \usr\ov\help\$LANG.
  - **ovwHelpFile**: The path of an ovhelp help file relative to the registered help directory, which is assumed to be helpRequest. If the application has not registered a help directory, the path is assumed to be relative to the NetView help directory, \usr\ov\help\$LANG.

Return Values

If successful, OVwShowHelp returns 0 (zero). If unsuccessful, −1 (negative one) is returned.

Error Codes

OVwShowHelp sets the error code value that OVwError returns. The following list describes the possible errors:

- **[OVw_APP_NOT_FOUND]**: The NetView help system application, ovhelp, is not running.
- **[OVw_CONNECTION_LOST]**: The connection to the NetView program was lost.
- **[OVw_OUT_OF_MEMORY]**: A memory allocation failure occurred.
The GUI API has not been initialized with OVwInit.

Implementation Specifics
OVwShowHelp supports single-byte and multibyte character code sets.

Libraries
When compiling a program that uses OVwShowHelp, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Files
OV\ovw.h

Implementation Specifics
The environment variable LANG determines the language in which messages are displayed. If LANG is not specified or is set to the empty string, a default of C is used instead of LANG. If any internationalization variable contains a setting that is not valid, the OVwShowHelp command functions as though all internationalization variables are set to C.

The ovwshowhelp command supports single-byte character code sets.

Related Information
- See “netview” on page 65
- See “OVwError” on page 520
- See “OVwInit” on page 564
- See “OVwApiIntro” on page 395
OVwSubmapCloseCB

Purpose

Functions as a callback for a submap-close event.

Syntax

```c
#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" on page 583.) 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 multibyte character code sets.

Libraries

When compiling a program that uses OVwSubmapCloseCB, link to the following libraries:
- `libovw.lib`
- `libov.lib`
- `libntl.lib`

Related Information

- See "netview" on page 65
- See "OVwAddCallback" on page 383
- See "OVwSubmapOpenCB" on page 649
- See "OVwApiIntro" on page 395
- See "OVwMapCloseCB" on page 583
**OVwSubmapOpenCB**

**Purpose**

Functions as a callback for a submap open event.

**Syntax**

```c
#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 multibyte character code sets.

**Libraries**

When compiling a program that uses OVwSubmapOpenCB, link to the following libraries:
- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntl.lib`

**Related Information**

- See "ovwSubmapOpen" on page 648
- See "OVwAddCallback" on page 382
- See "OVwDisplaySubmap" on page 515
- See "OVwSubmapCloseCB" on page 648
- See "OVwApiIntro" on page 395
- See "OVwGetMapInfo" on page 544
**OVwUserSubmapCreateCB**

**Purpose**

Functions as a callback for a user submap create event.

**Syntax**

```c
#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 on page 628](#) 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 on which the user is trying to open 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:

```c
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 multibyte character code sets.

Libraries

When compiling a program that uses OVwUserSubmapCreateCB, link to the following libraries:
- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libntl.lib

Related Information

- See "netview" on page 62.
- See "OVwAckUserSubmapCreate" on page 375.
- See "OVwAddCallback" on page 382.
- See "OVwApiIntro" on page 395.
- See "OVwSetSymbolApp" on page 628.
- See "OVwGetMapInfo" on page 544.
OVwVerifyAdd

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

The OVwQueryAddSymbolCB and OVwConfirmAddSymbolCB 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" on page 395 for an overview of the GUI API, including the role of the asynchronous NetView events.

An application that should 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 program will send an OVwQueryAddSymbol event to the appropriate application. The NetView 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 alphabetic 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 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 that the symbol is placed in the user plane. If there is a problem, 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, 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 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 reverify the information on the screen before pressing 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 must recheck them when the OVwConfirmAddSymbol event is received. If one or more of the values is incorrect, 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 on page 591 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 program to send an ovwQueryAddSymbol event before the user presses the Verify button. The difference is the action that triggers the NetView program to send the event. If a dialog box has the InitialVerify option set, the NetView 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 program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, the NetView program sends the ovwQueryAddSymbol event whenever the user exits that field. This function 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. If the value of this field is TRUE, the symbol is valid on the application plane. If not, the object will be added to the user plane. The appPlane field value should be FALSE is the verified field value 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 ('/n') 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, −1 (negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Examples
The following example illustrates typical callback routines:
Implementation Specifics

OVwVerifyAdd and its related functions support single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwVerifyAdd or one of its related functions, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libnt1.lib`

Related Information

- See "virtual" on page 65.
- See "OVwAddCallback" on page 383.
- See "OVwCreateSubmap" on page 445.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
• See "OVwGetMapInfo" on page 544.
 OVwVerifyAppConfigChange

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 on page 395](#) for an overview of the GUI API, including the role of the asynchronous NetView 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 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 on page 591](#) 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 and 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 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 that is not valid value is an integer field that is specified out of range.

**Note:** When the user selects File..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.
All fields in the dialog box, with their values, are in the \textit{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 \texttt{OVwVerifyAppConfigChange} call. If no change is necessary, the list is passed back unchanged.

If the field values are valid, the callback routine responds by calling \texttt{OVwVerifyAppConfigChange} with the verified parameter set to \texttt{TRUE}. The graphical interface enables the OK button and the user can commit the changes. If there is a problem with one or more of the fields, the callback routine calls \texttt{OVwVerifyAppConfigChange}. The verified parameter is set to \texttt{FALSE} and the \textit{errorMsg} is set to an appropriate message. In this case, the graphical interface will keep the OK button disabled and will display the message so that the user can fix the problem and reverify 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 program sends an \texttt{ovwConfirmAppConfigChange} event to the application. The \texttt{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* on page 591] for a more detailed description. These options are useful for making the dialog box interactions easier to use. They are similar in that each causes the NetView program to send an \texttt{ovwQueryAppConfigChange} event before the user presses the Verify button. The difference is the action that triggers the NetView program to send the event.

If a dialog box has the InitialVerify option set, the NetView program sends the \texttt{ovwQueryAppConfigChange} event when the user opens the application-specific describe dialog box but before it is actually displayed. This enables the application to fill in some default values for the fields by setting the values of the fields in \textit{ConfigParams} and \textit{dialogBoxFields}. After the application calls \texttt{OVwVerifyAppConfigChange}, the NetView program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, then the NetView program sends the \texttt{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 “\texttt{n}” 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 \texttt{OVwGetMapInfo} or saved from the \texttt{ovwMapOpen} event using \texttt{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, −1 (negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
   A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
   The GUI API has not been initialized with OVwInit.

Examples
   See "OVwVerifyAdd" on page 652 for an example with event callbacks. 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, link to the following libraries:
   • /usr/ov/lib/libovw.lib
   • /usr/ov/lib/libov.lib
   • /usr/ov/lib/libntl.lib

Related Information
   • See "netview" on page 65
   • See "OVwAddCallback" on page 383.
   • See "OVwError" on page 520.
   • See "OVwGetAppConfigValues" on page 527.
   • See "OVwInit" on page 564.
   • See "OVwApiIntro" on page 395.
   • See "OVwRegIntro" on page 591.
   • See "OVwGetMapInfo" on page 544.
**OVwVerifyConnect**

**Purpose**

Validates the user-selected connect operation for two symbols.

**Related Functions**

- OVwQueryConnectSymbolsCB
- OVwConfirmConnectSymbolsCB

**Syntax**

```c
#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 on page 393](#) for an overview of the GUI API, including the role of the asynchronous NetView 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 on page 591](#) 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 program will send an ovwQueryConnectSymbols event to the appropriate application. The NetView 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, the graphical interface will hang.

If OVwVerifyConnect is called with the verified parameter set to TRUE, 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 program sends an ovwConfirmConnectSymbols event to the appropriate applications.

When the ovwConfirmConnectSymbols is received, the application must ensure 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 on page 591 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 program to send an ovwQueryConnectSymbols event before the user presses the Verify button. The difference is the action that triggers the NetView program to send the event. If a dialog box has the InitialVerify option set, the NetView 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 program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, the NetView 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 \n 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 the first object

object2
Specifies a pointer to the ObjectInfo structure for the second object

appPlane
Specifies whether the object can be added to the application plane of the submap identified by submap. 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 can commit the changes; otherwise, changes are not valid.

Return Values
If successful, OVwVerifyConnect returns 0 (zero). If unsuccessful, −1 (negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Examples
See the example for OVwQueryAddSymbol in "OVwVerifyAdd" on page 652 for an example with event callbacks. The interaction model for OVwQueryAddSymbolCB is similar to that for OVwQueryConnectSymbolsCB.
Implementation Specifics

OVwVerifyConnect supports single-byte and multibyte character code sets.

Libraries

When compiling a program that uses OVwVerifyConnect and its related functions, link to the following libraries:

- `\usr\ov\lib\libovw.lib`
- `\usr\ov\lib\libov.lib`
- `\usr\ov\lib\libntli.lib`

Related Information

- See "netview" on page 65,
- See "OVwAddCallback" on page 383,
- See "OVwError" on page 520,
- See "OVwInit" on page 564,
- See "OVwApilntro" on page 395,
- See "OVwBglntro" on page 591,
- See "OVwGetMapInfo" on page 544.
OVwVerifyDeleteSymbol

Purpose
Validates the deletion of symbols.

Related Functions
OVwQueryDeleteSymbolsCB
OVwConfirmDeleteSymbolsCB

Syntax

```c
#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 when a delete operation is performed. See [OVwApiIntro](#) on page 395 for an overview of the GUI API including the role of the asynchronous NetView events.

An application should 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 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, 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 program. Once the NetView program gets the list back, it will selectively delete the symbols. The symbols for which the application set verified to TRUE are deleted but the symbol for which verified was FALSE are not. A dialog box for the symbols that were not deleted is then displayed that informs 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 because other applications may depend on their existence.

For the set of deleted symbols, the NetView program sends an ovwConfirmDeleteSymbols event. When the application receives the
ovwConfirmDeleteSymbols event, it must delete that symbol from its internal structures because it has been removed from the map.

If the last symbol for a given object is deleted, the graphical interface deletes that object and sends an ovwConfirmDeleteObject event. See "OVwConfirmDeleteObjectsCB" on page 423 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.

classList
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, −1 (negative one) is returned.

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 program was lost.

[OVw_OUT_OF_MEMORY]
A memory allocation failure occurred.

[OVw_OVW_NOT_INITIALIZED]
The GUI API has not been initialized with OVwInit.

Examples
See the example in "OVwVerifyAdd" on page 653 for a sample interaction.

Implementation Specifics
OVwVerifyDeleteSymbol and its related functions support single-byte and multibyte character code sets.
Libraries

When compiling a program that uses OVwVerifyDeleteSymbol or one of its related functions, link to the following libraries:

- \usr\ov\lib\libovw.lib
- \usr\ov\lib\libov.lib
- \usr\ov\lib\libnt1.lib

Related Information

- See "netview" on page 65.
- See "OVwAddCallback" on page 382.
- See "OVwConfirmDeleteObjectsCB" on page 423.
- See "OVwError" on page 520.
- See "OVwInit" on page 564.
- See "OVwApiIntro" on page 395.
- See "OVwRegIntro" on page 591.
- See "OVwGetMapInfo" on page 544.
OVwVerifyDescribeChange

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 on page 395] for an overview of the GUI API including the role of the asynchronous NetView events.

An application that should 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 on page 591] for more information about this file.

The application's Object Description dialog box presents several buttons on the bottom of the box: OK, Verify, and Cancel. When the box is first opened, the OK button is grayed out and 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 program sends 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, the application must call OVwVerifyDescribeChange with the verified parameter set to TRUE. If any of the fields are not correct, the application must call OVwVerifyDescribeChange with the verified parameter set to FALSE. If the application fails to call OVwVerifyDescribeChange, the graphical interface will be suspended.
If OVwVerifyDescribeChange is called with the verified parameter set to TRUE, 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 program sends an ovwConfirmDescribeChange event to the appropriate applications.

When the ovwConfirmDescribe is received, the application must update internal structures or its database, or both because the NetView 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" on page 591 for a more detailed description. These options are useful for making the dialog box interactions easier to use. They are similar in nature. They each cause the NetView program to send an ovwQueryDescribe event before the user presses the Verify button. The difference is the action that triggers the NetView program to send the event. If a dialog box has the InitialVerify option set, the NetView program will send the ovwQueryDescribe event when the user opens the application-specific Object Description dialog box, but before it is 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 program fills in the field values and displays the dialog box.

If a field has the ImmediateVerify option set, the NetView program sends the ovwQueryDescribe 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

- **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, −1
   (negative one) is returned.

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 program was lost.
   [OVw_OUT_OF_MEMORY]
      A memory allocation failure occurred.
   [OVw_OVW_NOT_INITIALIZED]
      The GUI API has not been initialized with OVwInit.

Examples
   See the example in "OVwVerifyAdd" on page 652 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
   multibyte character code sets.

Libraries
   When compiling a program that uses OVwVerifyDescribeChange or one of its
   related functions, link to the following libraries:
      \usr\ov\lib\libovw.lib
      \usr\ov\lib\libov.lib
      \usr\ov\lib\libntl.lib

Related Information
   • See "netview" on page 65.
   • See "OVwAddCallback" on page 382.
   • See "OVwError" on page 520.
   • See "OVwInit" on page 564.
   • See "OVwApiIntro" on page 395.
   • See "OVwRegIntro" on page 591.
   • See "OVwGetMapInfo" on page 544.
**polld**

**Purpose**

The polld daemon monitors the state of other daemons and restarts them as needed.

**Syntax**

`polld -spmd`

**Description**

The polld daemon monitors the state of daemons and restarts them as needed. The auto-start properties of the daemons monitored by the polld daemon are configured in the tab page of the daemon in the Polling Options dialog box.

The Polling Options GUI shows two groups of daemons: Minimum and Web. The minimum daemons are: ovwdb, ovtopmd, trapd, netmon, and nvcold.

The web daemons are: webserver, mibserver, and netviewd.

**Related Information**

- mibserver not done yet
- See *netmon* on page 59.
- See *netviewd* on page 68.
- not on NT
- See OvsPMD_API on page 356
- See *ovtopmd* on page 363
- See *ovwdb* on page 464
- See *trapd* on page 700
- webserver not done yet
QuickTest

Purpose

Ping tool that will poll all managed interfaces for a node.

Syntax

quicktest -n nodename [-z] [-P] [-F]

where

-n nodename specifies the node to be polled and is required.

Description

QuickTest is an enhanced Ping tool that will poll all managed interfaces for a node. It provides the ability to quickly determine the complete status of a node. QuickTest will display the ifAdminStatus and ifOperStatus of each interface for nodes that use SNMP to poll for status. For all other nodes, QuickTest will display the Ping result for each interface. If the status of an interface has changed, the map will be updated and an event will be sent.

QuickTest is available from the NetView console and NetView Web client by selecting Test...QuickTest or Test...QuickTest Critical. You must have one or more nodes selected to access the QuickTest menus. The Test...QuickTest Critical menu item will only poll interfaces that are currently marked as down. The Test...QuickTest menu item will poll all interfaces, regardless of their current status.

h

Flags

-z Only poll down interfaces.

-P Use ping instead of SNMP requests to poll SNMP Status nodes.

-F Send an event for all polled interfaces, regardless of whether there is a status change. Normally, an event is only sent when the status has changed.

Related Information

- See netmon on page 59.
- See netview on page 65.
- See nmdemandpoll on page 69.
**rnetstat**

**Purpose**

Lists the network information of a remote SNMP node.

**Syntax**

```
rnetstat [-a|-A|-e|-i|-I|-r|-S|-u] [-t timeout] [-R retries] [node]
```

**Description**

The `rnetstat` command lists the contents of various network-related data structures on `node`. The information is displayed in various formats, depending on the options specified.

By default, the connection table appears; this table includes the local and remote address and port number, and the TCP state.

The node must be a node supporting SNMP; the default node is the local node.

Address formats have the form host.port. When known, the host and network addresses are displayed. If a symbolic name for an address is unknown or if the `-n` option is specified, the address is printed in Internet dot format. Unspecified and wildcard addresses and ports are displayed as an asterisk (*).

**Flags**

- `-a` Shows all TCP and UDP table entries; generally, sockets used by server processes are not shown.
- `-A` Shows the cache entries; these entries include the address to physical (link level) address mappings.
- `-e` Shows ethernet (physical or link level) statistics, including interface name, number of frames received and transmitted, and numbers of input errors, output collisions, deferred packets, and framing errors. If the `-l` option is specified, output is in a listing format that includes additional interface-specific information.
- `-i` Shows the network interfaces and their associated properties and statistics; includes interface name, input packets, input errors, output packets, and output errors.
- `-l` Shows the IP address table. Includes interface name, IP address, network mask, network address, and physical address.
- `-l` Format the output as a long list, rather than as the default table format. When the `-l` flag is specified, more information is provided, if available. You can format the output as a long list only when the `-l` flag is combined with the `-e` flag.
- `-n` Shows network addresses as numbers rather than listing them symbolically.
- `-N` Specifies not to truncate names. By default, names are truncated to align the output columns.
- `-r` Shows the IP routing table; includes the destination address, next node in the route, type of routing entry, and name of the interface used for the route. The type of the route is either direct or remote. A direct connection implies that no remote gateway is explicitly used to reach the destination. A remote connection implies a route through a non-local gateway.
-S  Shows the services table. This table is the opposite of the default TCP connection table, and shows the TCP and UDP table entries that match services. For each entry the protocol (TCP or UDP), port number, and service name is displayed.

-u  Lists the UDP table (local address and port information).

-c  community
    Overrides the community name as configured in \usr\ov\conf\ovsnmp.conf) to community. See \ovsnmp.conf\ on page 304.

-t  timeout
    Overrides the default time-out, in tenths of a second.

-R  retries
    Overrides the number of SNMP retries.

Implementation Specifics

The environment variable LANG determines the language in which messages are displayed. If LANG is not specified or is set to the empty string, a default of “C” is used instead of LANG. If any internationalization variable contains a setting that is not valid, rnetstat behaves as if all internationalization variables are set to “C.”

The rnetstat routine supports single-byte character code sets.

Dependencies

The rnetstat command uses SNMP. The target node must support SNMP.

Files

\usr\ov\conf\ovsnmp.conf
    SNMP community names

Related Information

In this document:
    See \ovsnmp.conf\ on page 304.

Request for comments:
    \usr\ov\doc\ (for related RFC’s)
Schema for NetView Databases

Purpose

Provides access to structured query language (SQL) databases for reports and queries.

Description

The NetView program stores event, SnmpCollect, and topology information in SQL databases. You can query databases and write reports using the information in the databases.

The NetView program can use Microsoft Access or Microsoft SQL Server databases. The type of database is determined at installation. If SQL Server is installed, the NetView installation creates the NetViewDev device name and the NetViewDB database to store the NetView schema.

The Schema for the Event System is:

<table>
<thead>
<tr>
<th>Table</th>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterpriseName</td>
<td>enterpriseOID</td>
<td>char(30)</td>
</tr>
<tr>
<td></td>
<td>enterpriseName</td>
<td>char(50)</td>
</tr>
<tr>
<td>eventOwners</td>
<td>ownerID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>ownerName</td>
<td>char(80)</td>
</tr>
<tr>
<td>events</td>
<td>enterpriseOID</td>
<td>char(30)</td>
</tr>
<tr>
<td></td>
<td>generic</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>specific</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>node</td>
<td>char(60)</td>
</tr>
<tr>
<td></td>
<td>description</td>
<td>char(60)</td>
</tr>
<tr>
<td></td>
<td>severity</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>category</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>hasNote</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>eventID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>ownerID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>int</td>
</tr>
<tr>
<td>moreDescription</td>
<td>eventID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>moreDescription</td>
<td>yes, dups ok</td>
</tr>
<tr>
<td>Notes</td>
<td>eventID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>theNote</td>
<td>char(255)</td>
</tr>
<tr>
<td>trapNames</td>
<td>enterpriseOID</td>
<td>char(30)</td>
</tr>
<tr>
<td></td>
<td>specific</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>generic</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>enterpriseName</td>
<td>char(50)</td>
</tr>
<tr>
<td></td>
<td>specificName</td>
<td>char(20)</td>
</tr>
</tbody>
</table>

The Schema for snmpCollect is:

<table>
<thead>
<tr>
<th>Table</th>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mibLookup</td>
<td>mibID</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>mibName</td>
<td>char(80)</td>
</tr>
</tbody>
</table>
### SnmpcollectData

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mibObject</td>
<td>int</td>
</tr>
<tr>
<td>instance</td>
<td>int</td>
</tr>
<tr>
<td>ipAddress</td>
<td>int</td>
</tr>
<tr>
<td>startTime</td>
<td>int</td>
</tr>
<tr>
<td>endTime</td>
<td>int</td>
</tr>
<tr>
<td>value</td>
<td>double</td>
</tr>
<tr>
<td>stationID</td>
<td>int</td>
</tr>
<tr>
<td>theKey</td>
<td>int</td>
</tr>
</tbody>
</table>

### SnmpcollectStation

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stationID</td>
<td>int</td>
</tr>
<tr>
<td>station</td>
<td>char(60)</td>
</tr>
</tbody>
</table>

The Schema for Topology (topo2sql dump utility) is:

### Networks

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectID</td>
<td>int(20)</td>
</tr>
<tr>
<td>IPAddress</td>
<td>char(20)</td>
</tr>
<tr>
<td>status</td>
<td>char(16)</td>
</tr>
</tbody>
</table>

### Nodes

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectID</td>
<td>int</td>
</tr>
<tr>
<td>objectID2</td>
<td>int</td>
</tr>
<tr>
<td>Name</td>
<td>char(50)</td>
</tr>
<tr>
<td>status</td>
<td>char(16)</td>
</tr>
<tr>
<td>IPAddress</td>
<td>char(30)</td>
</tr>
<tr>
<td>MACAddress</td>
<td>char(20)</td>
</tr>
</tbody>
</table>

### Segments

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectID</td>
<td>int</td>
</tr>
<tr>
<td>IPAddress</td>
<td>char(30)</td>
</tr>
<tr>
<td>status</td>
<td>char(16)</td>
</tr>
</tbody>
</table>
smartsetmaint

Purpose

Enables you to save and restore SmartSet.

Syntax

smartsetmaint -d [filename]
smartsetmaint -l [filename]
smartsetmaint -r [SmartSet Name]
smartsetmaint -o [SmartSetName object-id]
smartsetmaint -v

Parameters

-d [filename]
  Saves SmartSets from the object database into a file. If you omit a file name, the default file name is '/usr/ov/conf/smartset.conf'.

-l [filename]
  Restores SmartSets into the object database. Make sure that the nvcold daemon is not running when you use -l.

-r
  Removes a SmartSet entry from the database. Make sure the nvcold daemon is not running when you use -r.

-o [SmartSetName object-id]
  Removes a SmartSet entry from the database. Make sure that the nvcold daemon is not running when you use -o.

-v
  Displays SmartSet entries in the object database. Use this option to find the Object ID of a SmartSet.

Related Information

See "smartsetutil" on page 673.
smartsetutil

Purpose
The smartsetutil command enables you to create, delete, or list SmartSets.

Syntax
smartsetutil<option letter> <required parameters>

Parameters
L Displays all SmartSets.
G Displays all SmartSets and their rules.
g SmartSetName
Displays the description and rule for the SmartSet. For example, “smartsetutil
g Routers” displays a description of the Routers SmartSet and the rule it uses
to include objects.
I SmartSetName
Displays members of a SmartSet. For example, “smartsetutil I Routers”
displays the objects that match the “isRouter” = “TRUE” rule.
e SmartSetRule
Displays the objects that match the rule. For example, “smartsetutil e "isNode = True"” displays all objects that are nodes.
p SmartSetRule
Displays the objects that match the rule along with the object’s properties. For example, ”smartsetutil e “isRouter = True”.”
a SmartSetName SmartSetDesc SmartSetRule [force]
Creates a new SmartSet. For example, ”smartsetutil a mybridges MyNetBridges “isBridge = True” 1.” You can omit force.
D SmartSetName [force]
Deletes a single SmartSet. For example, ”smartsetutil D mybridges”. You can
omit force.
X Deletes all SmartSets.
d SmartSetName SmartSetDesc
Changes the description of a SmartSet. For example, ”smartsetutil d
mybridges “Bridges in Segment11””.
r SmartSetName SmartSetRule
Changes the rule of a SmartSet. For example, ”smartsetutil r mybridges
“isIPRouter = True”.”
m SmartSetName SmartSetDesc SmartSetRule
Changes the description and rule of a SmartSet. For example, ”smartsetutil m
mybridges “Bridges that are IP Routers” “isIPRouter = True”.”
s oid
Displays SmartSets that contain the object. For example, ”smartsetutil s 528”.
S Displays the name, description, rule, and current evaluation for all SmartSets
information.
**u SmartSetName1 SmartSetName2**
Displays the union of the members in both SmartSets. For example, "smartsetutil u Routers MyWorkStations" displays the objects belonging to Routers and MyWorkStations.

**c SmartSetName oid**
Displays the union of the members in the SmartSet and the object matching the object ID. For example, "smartsetutil c Routers 606" displays the objects belonging to Routers and the object matching the 606 ID.

**i SmartSetName1 SmartSetName2**
Displays the intersection of the members. These are members that both SmartSets have in common. For example, assume that one device is both a router and a workstation, the "smartsetutil i Routers MyWorkStations" command displays one member as common to both.

**t SmartSetName oid**
Displays the intersection of a SmartSet and an object ID. For example, "smartsetutil t Routers 711".

The following is a review of the required/additional parameters:
- SmartSetName is the name of a SmartSet
- SmartSetDesc is the description of a SmartSet
- SmartSetRule is the conditions that objects meet to be included in a SmartSet
- Force is 0 or 1 that forces the operation to occur even if dependencies exist.
- OID is the object ID of an object from the object database

**Related Information**
See [smartsetmaint on page 677](#)
snmpCollect

Purpose

Collects, compares, and stores SNMP MIB values.

Syntax

```
```

Description

The snmpCollect daemon is configured through the Tools...MIB...Collect Data operation of the graphical interface. The snmpCollect daemon collects values from network nodes at regular intervals and, if so configured, stores them to a file. The snmpCollect daemon can also compare the collected values with user-defined threshold values and generate events when the thresholds are exceeded.

When netmon discovers a new node, snmpCollect checks to see if the new node’s address is contained within any of the address pattern-matching character conditions specified to determine if any collections should be automatically started on the new node.

If a node does not reply because it or its agent is down, snmpCollect defers requests on the node until the defer time has expired, or a demand poll or object ID changed event occurs. The default defer time is 60 minutes. At snmpCollect invocation, the -D deferMinutes option allows you to specify a different defer time.

Collections can also be done by a proxy agent node. The proxy target must be included in the map, and the appropriate entries added in /usr/ov/conf/ovsnmp.conf, which is modified by the Options..SNMP operation of the graphical interface.

If you specify a tracefile then starting snmpCollect, tracing is automatically turned on. The snmpCollect routine automatically puts itself in the background when it is started, whether or not a tracefile is specified.

Note: The snmpCollect daemon requires that the trapd daemon be running so it can listen for newly discovered nodes and send threshold events.

You can only collect data from managed nodes. If you unmanage the node using the NetView program, collection ceases until you manage the node again.

Flags

The snmpCollect daemon accepts the following options. The first three options can be used only when starting up snmpCollect. The last four options can be used only to send signals to snmpCollect once it is running.

-D deferMinutes

At snmpCollect startup, the -D deferMinutes option allows you to specify a different defer time. This startup option generates an error message if snmpCollect is already running. If a node does not reply (because it or its agent is down), snmpCollect defers requests on the node until the defer time has expired, or a “demand poll” or “object id changed” event occurs from netmon. The default defer time is 60 minutes.
-t *traceFile*
  The -t *traceFile* option allows you to specify a different *traceFile* destination. This option generates an error message if snmpCollect is already running. When tracing is turned on, snmpCollect writes trace and status information to the default file `\usr\ov\log\snmpCol.trace`.

-u Collects on nodes even if the nodes are unmanaged. Without this option, snmpCollect does not collect on nodes that are not managed.

-S Appends the configuration of snmpCollect to the trace file, regardless of the current state of tracing.

-T Specifies toggle tracing. By default snmpCollect starts with tracing off unless *traceFile* is specified with the -t option at snmpCollect invocation.

-V Specifies toggle verbose tracing. If tracing is not currently on, the next time tracing is turned on it will be verbose.

-K Kills snmpCollect to stop data collection.

**Storing Data**
If data storing is enabled, collected data is stored in the directory `\usr\ov\databases\snmpCollect`. For each variable, snmpCollect creates the following two files:

- A binary data file containing the data collected for that variable. The binary file has the same name as the label specified in the MIB Data Collection menu.
- A separate ID file containing the units, syntax, and object for the variable. This file has the same name as the binary data file, except there is a ! appended to the filename.

The contents of the binary data file consists of quadruplets of startTime, endTime, IPaddress, and value. The startTime, endTime, and IPaddress are 8-byte, unsigned long values. The collected value is an 8-byte, double value. To convert the binary data to human readable format use snmpColDump or write your own program.

If the binary data file is removed at any time, snmpCollect creates a new one at the next collection. The binary files grow unbounded, so it is advisable to remove them periodically, either manually or by using the Windows NT at command. Be sure not to remove the ID files, that is, filenames with a ! at the end of them. To delete earlier data, use the -r option to snmpColDump. See "snmpodump" on page 687.

The trace file also grows without bounds. It can also be removed, but tracing must be toggled twice to stop and restart it.

**Threshold Events:** If a threshold is enabled (see "thresh" on page 696) and the collected variable value exceeds the threshold, an trap is generated. A rearm value can also be defined to eliminate the “noise” of a variable value bouncing around the threshold. This rearm value indicates how far below the threshold value a collected value must fall before another threshold event can be generated. When a collected value falls below or is equal to the rearm value, a rearm event is sent.

The action taken on receipt of a threshold event can be modified through the Options..Events..Trap Settings operation of the graphical interface. Select the IBM AIX enterprise, the generic trap number 6, and the specific trap number 58720263, then specify an appropriate trap action. The following information is passed in the threshold trap variables:

<table>
<thead>
<tr>
<th>Variable Number</th>
<th>Content</th>
</tr>
</thead>
</table>

*Variable Number*  
*Content*
2 Host name

3 The string “MIB_LABEL threshold exceeded (>THRESHOLD_VALUE): COLLECTED_VALUE”

4 The object ID of the MIB variable

Similarly, the trap sent on a rearm is the IBM AIX enterprise, generic trap number 6, and specific trap number 58720264, with the following information sent as trap variables:

<table>
<thead>
<tr>
<th>Variable Number</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Host name</td>
<td></td>
</tr>
<tr>
<td>3 The string “MIB_LABEL threshold rearmed (&lt;=REARM_VALUE): COLLECTED_VALUE. Sampled high of HIGH_VALUE at TIME”</td>
<td>The HIGH_VALUE is the highest sampled value since the threshold was detected.</td>
</tr>
<tr>
<td>4 The object ID of the MIB variable</td>
<td></td>
</tr>
</tbody>
</table>

**Files**

\`\texttt{\textasciitilde ov\textbackslash databases\textbackslash snmpCollect\*}
\`\texttt{\textasciitilde ov\textbackslash databases\textbackslash snmpCollect\*[!]}
\`\texttt{\textasciitilde ov\textbackslash log\textbackslash snmpCol.trace}
\`\texttt{\textasciitilde ov\textbackslash conf\textbackslash collectProxyMap}
\`\texttt{\textasciitilde ov\textbackslash conf\textbackslash snmpCol.conf}
\`\texttt{\textasciitilde ov\textbackslash conf\textbackslash mibExpr.conf}
\`\texttt{\textasciitilde ov\textbackslash pids\textbackslash nvcollection_pid}
\`\texttt{\textasciitilde ov\textbackslash conf\textbackslash ovsnmp.conf}

**Related Information**

- See ["netview" on page 65](#).
- See ["snmpodump" on page 687](#).
- See ["ovstart" on page 358](#).
- See ["trapd" on page 700](#).
- See ["mibExpr.conf" on page 51](#).
- See ["ovsnmp.conf" on page 304](#).
snmpget

Purpose
Queries a node using the SNMP Get Request.

Syntax
```
snmpget [-r retries] [-p port] [-c community] node variable [...]
```

Description
The snmpget command uses the SNMP getrequest to query a node for information. The snmpget command accepts 1-to-20 fully qualified object IDs as arguments.

Each variable has the format of A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation. The default variable prefix is .iso.org.dod.internet.mgmt.mib. You must include the instance ID (generally .0) at the end of the object ID.

If the target node has an error processing the request packet, an error packet returns, and a message is displayed to identify how the request was incorrectly formatted.

Flags
The default values for the option is determined by the configuration in \usr\ov\conf\ovsnmp.conf. See "ovsnmp.conf" on page 304.

-? Lists the valid variable types and the usage message.

-d Dumps to standard output all SNMP packets in a hexadecimal and decoded ASN.1 format.

-r Overrides the default number of retries to retries. An exponential backoff algorithm is used, so if timeout was 10 (1 second) and retries was 3, the first timeout would occur at 1 second, the second (first retry) at 2 seconds, the third (second retry) at 4 seconds, and the last (third retry) at 8 seconds. It would take 15 seconds before the command gave up.

-p Overrides the default port for sending/receiving to port.

-c Overrides the SNMP community name (as configured in \usr\ov\conf\ovsnmp.conf) to community. See "ovsnmp.conf" on page 304.

If all arguments are formatted correctly, a message is displayed indicating that the SNMP trap was sent to the remote node.

Examples
You can request the system description and system uptime for testnode using the SNMP community name public by entering the following command:
```
snmpget -c public testnode system.sysDescr.0 system.sysUpTime.0
```

The following example depicts the output for the above command:
```
system.sysDescr.0 : DISPLAY STRING- (ascii): IBM RS/6000
Machine Type: 0x0014 Processor id:000001121400
The Base Operating System AIX version: 03.02.0000.0000
TCP/IP Applications version: 03.02.0000.0000
system.sysUpTime.0 : Timeticks: (25238044) 2 days, 22:06:20.44
```
Files

\usr\ov\conf\ovsnmp.conf

Related Information

- See "snmpnext" on page 685
- See "snmpset" on page 690
- See "mvsnpmptrap" on page 270
- See "snmpwalk" on page 692
- See "ovsnmp.conf" on page 304

Request for comments

- RFC 1109
- RFC 1155
- RFC 1156
- RFC 1157
- RFC 1158
snmpnext

Purpose
Queries a node using the SNMP GetNext Request.

Syntax

```
snmpnext [-r retries] [-p port] [-c community] node variable [...]
```

Description
The snmpnext command uses the SNMP getnext Request to query a node for information. The snmpnext command accepts 1-to-20 fully-qualified object IDs as arguments. For each variable, the SNMP variable that is lexicographically next in the target node’s MIB is returned.

Each variable has the format A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation or group.variable notation. The default variable prefix is .iso.org.dod.internet.mgmt.mib. The default values for the options is determined by the configuration in \usr\ov\conf\ovsnmp.conf. See "ovsnmp.conf" on page 304.

If the target node has an error processing the request packet, an error packet returns, and a message is displayed to identify how the request was incorrectly formatted.

Flags

-? Lists the valid variable types and the usage message.

-d Dumps to standard output all SNMP packets in a hexadecimal and decoded ASN.1 format.

-r Overrides the default number of retries to retries. An exponential backoff algorithm is used, so if timeout was 10 (1 second) and retries was 3, the first timeout would occur at 1 second, the second (first retry) at 2 seconds, the third (second retry) at 4 seconds, and the last (third retry) at 8 seconds. It would take 15 seconds before the command gave up.

-p Overrides the default port for sending or receiving to port.

-c Overrides the SNMP community name (as configured in \usr\ov\conf\ovsnmp.conf) to community. See "ovsnmp.conf" on page 304.

If all arguments are formatted correctly, a message is displayed indicating that the SNMP trap was sent to the remote node.

Examples

You can request the system description and system uptime for testnode using the SNMP community name public by entering the following command:

```
snmpnext -c public testnode system.sysDescr system.sysUpTime
```

The following example depicts the output for the above command:
system.sysDescr.0 : DISPLAY STRING-(ascii): IBM RS/6000
Machine Type: 0x0014 Processor id: 000001121400
The Base Operating System AIX version: 03.02.0000.0000
TCP/IP Applications version: 03.02.0000.0000
system.sysUpTime.0 : Timeticks: (25252514) 2 days, 22:08:45.14

Files

\usr\ov\conf\OVsnmp.conf

Related Information

- See "snmpget" on page 683.
- See "snmpset" on page 690.
- See "nvsnmtrap" on page 270.
- See "snmpwalk" on page 692.
- See "ovsnmp.conf" on page 304.

Request for comments

- \usr\ov\doc\* (for related RFC’s)
- RFC 1109
- RFC 1155
- RFC 1156
- RFC 1157
- RFC 1158
snmpodump

Purpose

Formats data collected by snmpCollect. It also supports deleting old data from the snmpCollect relational database and compressing existing data.

Syntax

snmpodump [-pPtT] [-m mibname] [-v value][-V value][-s{nNtTvVmM}] [-wn] [-hn]

snmpodump [-c] [gn][-Sn][-rn]

snmpodump [-xn]

Description

The snmpodump command formats data from the snmpcollect relational database into a readable and parseable format. It also provides the capability to purge old data from the database and compress existing data (which reduces storage space but provides less granularity).

The default operation of snmpodump is to format and output the collected SNMP data from the snmpcollect relational database. There are a series of flags that control how this is done. The -p, -P, -t, and -T switches control how the date is displayed for each record. The -m, -v, and -V flags control filtering of the data, displaying only those records matching either a specific MIB variable or value. The -s flag specifies how the data is to be sorted.

The snmpodump command can be used to delete old data, using the -x flag, leaving only the n latest data records. This operation is invoked by default by the schedule service to remove the 5000 oldest records once a day.

To compress the existing data in the snmpcollect database, use the -c flag. Each group of n records is analyzed, an average value is computed, the n records are deleted, and a new record with the average value is created. The value of n is controlled by the -g switch. For example, if you had collected data every 15 minutes and ran "snmpodump -c -g4" then each group of four 15 minute data points would be compressed into one 1 hour data point. These new data points are tagged as having a compression factor of four. By default, compression works only on uncompressed data. The -S switch allows snmpodump to perform additional re-compression of data. For example, "snmpodump -c -g6 -S4" will take all data with a compression factor of four and recompress them by an additional factor of six.

Flags

The following options may be used:

-c   Summarize (compress) the data.

-gn   Set the group size for summarization (turn every n record into one new record).

-Sn   Recompress all data with a compression factor of n.
-rn
Reschedule snmpodump data compression with the current settings every n
minutes; the valid range is 5..44640 minutes (up to 31 days maximum).

-p  Displays the collection period.
-P  Omit date in time display.
-t  Display the start time.
-T  Display the end time.

-mmibname
Only display data for the specified MIB, for example sysUpTime.

-mmibname.n
Only display data for instance n of the specified MIB variable.

-Vvalue
Show only MIB values that are greater than the specified value.

-s{nNtTvVmM}
Specify how the data should be sorted. To sort by IP address, specify "n" to
sort in ascending order, "N" in descending order. To sort by time, specify "t" to
sort in ascending order, "T" in descending order. To sort by value, specify "v" to
sort in ascending order, "V" in descending order. To sort by MIB variable,
specify "m" to sort in ascending order, "M" in descending order.

-wn
Display the last n minutes of data.

-hn
Show a maximum of n data points.

-xn
Purge the snmpCollect database leaving only the n most recent data entries.

Examples

snmpodump -t -T -v5 -sn
This example will display the start and end times for all records with values less
than 5.0 and sort them by IP address.

snmpodump -P -V5 -sV
This example will not display the collection period. It will show all records with
values greater than 5.0 and will sort them by value.

snmpodump -w5
This example will display all records collected in the last five minutes.

snmpodump -h50
This example will display the most recent 50 data points.

Files

\usr\ov\databases\snmpCollect\*
Binary data files
\usr\ov\databases\snmpCollect\*\[
ID files

688  Programmer’s Reference
Related Information

See [snmpCollect on page 680].
snmpset

Purpose

Issues an SNMP Set Request.

Syntax

```
snmpset [-d] [-t timeout] [-r retries] [-p port] [-c community] node variable type value [...]
```

Description

The snmpset command issues an SNMP set request to alter MIB objects on the remote `node` and returns the result of the Set Request. The snmpset command accepts 1-to-20 sets of the arguments `[variable type value]`.

Each `variable` has the format A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation and group.variable notation. The default variable prefix is `.iso.org.dod.internet.mgmt.mib`. You must include the instance ID (generally .0) at the end of the object ID.

Each `type` must be one of the following types: Integer, OctetStringHex, OctetStringOctal, OctetStringASCII (special cases of OctetString), ObjectIdentifier, Null, IpAddress, Counter, Gauge, TimeTicks, OpaqueHex, OpaqueOctal, or OpaqueASCII (special cases of Opaque). See RFC 1155 for a complete description of each type.

The `value` must be valid for the specified `type`. When using a `type` where a hexadecimal or octal value is needed (OctetStringHex, OctetStringOctal, OpaqueHex, OpaqueOctal), the `value` must have each byte fully defined. For example, fff (or 17377) is not allowed, whereas 0fff (or 017377) is allowed. For `type` Null, a `value` must be specified on the command line, but it is ignored when the request is created. A `value` must not occupy more than 256 bytes.

If the target node has an error processing the request packet, an error packet returns, and a message is displayed to identify how the request was incorrectly formatted.

Flags

The default values for the options is determined by the configuration in `\usr\ov\conf\ovsnmp.conf`. See `ovsnmp.conf` on page 304.

- `?-` Lists the valid variable types and the usage message.
- `-d` Dumps to standard output all SNMP packets in a hexadecimal and decoded ASN.1 format.
- `-r` Overrides the default number of retries to retries. An exponential backoff algorithm is used, so if timeout was 10 (1 second) and retries was 3, the first timeout would occur at 1 second, the second (first retry) at 2 seconds, the third (second retry) at 4 seconds, and the last (third retry) at 8 seconds. It would take 15 seconds before the command gave up.
- `-p` Overrides the default port for sending or receiving to `port`. 

Programmer’s Reference
Overrides the SNMP community name (as configured in \usr\ov\conf\ovsnmp.conf) to community. See ovsnmp.conf on page 304.

If all arguments are formatted correctly, a message is displayed indicating that the SNMP trap was sent to the remote node.

**Examples**

You can set the system contact to Bob Jones for testnode using the SNMP community name public entering the following command:

```bash
snmpset -c public testnode system.sysContact.0 OctetString "Bob Jones"
```

You can set the system contact to Bob Jones for the node testnode by entering the following command:

```bash
snmpset testnode system.sysContact.0 octetstring "Bob Jones"
```

The following example depicts the output for the previous command:

```
system.sysContact.0 : DISPLAY STRING- (ascii): Bob Jones
```

**Files**

\usr\ov\conf\ovsnmp.conf

**Related Information**

- See snmpset on page 683.
- See snmpnext on page 685.
- See nvsnmptrap on page 270.
- See snmpwalk on page 692.
- See ovsnmp.conf on page 304.

**Request for comments**

- \usr\ov\doc\* (for related RFC's)
- RFC 1109
- RFC 1155
- RFC 1156
- RFC 1157
- RFC 1158
snmpwalk

Purpose

Uses the SNMP Get Next Request to query a node for information.

Syntax

snmpwalk [-d] [-t timeout] [-r retries] [-p port] [-c community] node [variable]

Description

The snmpwalk command uses the SNMP getnext request to query node for information. The value of variable determines the portion of the object ID space that is searched using SNMP getnext requests. The first request is issued with the given variable. Each subsequent request works off of the variable returned with the previous request. The snmpwalk command terminates when the variable returned is no longer in the subtree indicated by variable. The default variable for snmpwalk is iso.org.

The variable has the format of A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation.

If there is an attempt to search beyond the end of the remote node’s MIB with snmpwalk or snmpnext, the message “End of MIB” is displayed.

Flags

The default values for the options are determined by the configuration in \usr\ov\conf\ovsnmp.conf. See ovsnmp.conf on page 304.

-? Lists the valid variable types and the usage message.

-d Dumps to standard output all SNMP packets in a hexadecimal and decoded ASN.1 format.

-t Overrides the default timeout with timeout, given in 1\10th of a second units.

-r Overrides the default number of retries to retries. An exponential back off algorithm is used, so if timeout was 10 (1 second) and retries was 3, the first timeout would occur at 1 second, the second (first retry) at 2 seconds, the third (second retry) at 4 seconds, and the last (third retry) at 8 seconds. It would take 15 seconds before the command gave up.

-p Overrides the default port for sending and receiving to port.

-c Overrides the SNMP community name (as configured in \usr\ov\conf\ovsnmp.conf ) to community. See ovsnmp.conf on page 304.

Examples

You can request the system subtree for node testnode by entering the following command:

snmpwalk -c public testnode system

Output for the above command typically resembles the following:

system.sysDescr.0 : DISPLAY STRING- (ascii): IBM RS/6000
Machine Type: 0x0014 Processor id: 000001121400
The Base Operating System AIX version: 03.02.0000.0000
TCPIP Applications version: 03.02.0000.0000
system.sysObjectID.0 : OBJECT IDENTIFIER:
.iso.org.dod.internet.private.enterprises.ibm.ibmAgents.1.2.1.1.2
system.sysUpTime.0 : Timeticks: (25352438) 2 days, 22:25:24.38
system.sysContact.0 : DISPLAY STRING- (ascii): John Doe
system.sysName.0 : DISPLAY STRING- (ascii): testnode
system.sysLocation.0 : DISPLAY STRING- (ascii): Somewhere in RTP
system.sysServices.0 : INTEGER: 72

You can set the system contact to John Doe for the node testnode by entering the following command:

```
snmpset testnode system.sysContact.0
tonetstring "John Doe"
```

The output for the above command is as follows:

```
system.sysContact.0 OCTET STRING-(ascii): John Doe
```

Files

\usr\ov\conf\ovsnmp.conf

Related Information

- See "ovsnmptrap" on page 270
- See "ovsnmp.conf" on page 304
- See "snmpget" on page 683
- See "snmpnext" on page 685
- See "snmpset" on page 690

Request for comments

\usr\ov\doc\* (for related RFC's)
spmsur.exe

Purpose

The spmsur.exe application can run as a daemon and launches a java application.

Syntax

```
spmsur.exe [-cd path] [-cp classpath] [-cls javaclass] [-a arguments]
```

Description

An instance of spmsur is launched by ovspmd.exe to start the webserver and mibserver daemons. The spmsur daemon acts as a wrapper to these Java applications, allowing ovspmd to exercise control over them.

Flags

- **-cd path**
  Specifies the directory the Java application is run from. For example:
  ```
  -cd \usr\OV\www
  ```

- **-cp**
  Specifies the java classpath to use. Webserver example:
  ```
  -cp \usr\OV\www\classes;\usr\OV\www\lib\il8n.jar
  \usr\OV\www\lib\xml.jar;\usr\OV\www\lib\log.jar
  \usr\OV\www\lib\nsvservlets.jar;
  \usr\OV\www\lib\servlet.jar;
  \usr\OV\www\lib\com.mortbay.jetty.jar
  ```
  Mibserver example:
  ```
  -cp \usr\OV\www\classes;\usr\OV\www\lib\il8n.jar
  \usr\OV\www\mibserver\mibserver.jar
  ```

- **-cls javaclass**
  Specifies the java class to run. For example:
  ```
  -cls com/mortbay/Jetty/Server
  or:
  -cls ibm/nways/jdm/NVJdmServer
  or:
  ```

- **-a**
  Specifies any arguments to the Java application. For example:
  ```
  \usr\OV\www\conf\JettyServer.prp
  ```

Related Information

- See "ovspmd" on page 354
- See "mibserver" on page 55
- See "webserver" on page 710
TESS

Purpose

To view events that are received by the management station.

Syntax

TESS

Description

The TESS command displays events received by the management system in the Event Browser. Events are stored in a database. You can view all the events received or view only events you are interested in by using filters.

The Event Browser can be started from the NetView Monitor menu, the Application Toolbar, or the NetView program group.

TESS enables you to specify filters to focus the event display you want to monitor. When you exit the event browser, it saves the last saved filter for use the next time you run the event browser.

Users can perform the following operations with TESS:

- Filter events
- Show the nodes on a map associated with the event that was generated
- Graph event traffic to create a graph of the overall event rate over time
- Generate statistics to display a breakdown by severity of all events that passed through a filter
- Add notes to events
- Acknowledge individual events by adding an owner name

Files

\us\lov\conf\trapd.conf

Describes the format in which to display SNMP traps

SQL Events Database

Stores events and traps

Related Information

- See "trapd" on page 700.
- See "trapd.conf" on page 703.
thresh

Purpose
Sets a threshold for MIB data collection.

Syntax

thresh [-D ] [-i ] instanceNum [-o ] objectIDName

Description
The thresh command sets up data collection configurations and store this configuration information in the \usr\ov\conf\snmpCol.conf file. If an entry already exists in the \usr\ov\conf\snmpCol.conf file with a MIB object ID and source name matching those you enter with the thresh command, that MIB object entry is updated with the new information.

You can collect data or monitor thresholds only on numeric MIB values, that is, on MIB objects that are defined as type Counter, Gauge, or Integer.

The graphical interface equivalent to the thresh command is the Tools..Collect Data operation.

Flags
-o objectIDName
Specifies the full MIB object ID mnemonic name. The MIB label and object ID are derived from the objectIDName.

-i instanceNum
Specifies on which instance of the MIB object you want to collect data. An instance is an internal counter on the system. For example, assume you have multiple disks on a node and the name of the MIB object is disks. The instance tells the data collector on which disk to collect data.

If the object on which you want to collect data does not support multiple instances, the instance is 0. If you have multiple instances of a MIB object on a device, for example, interface cards, and want to specify the first instance of the MIB object, type .1. To specify the instance of the second MIB object, type .2, and so forth.

-s SuspendRes
Specifies the status of data collection for the MIB object. Type the value R to start or resume collecting or type the value S to suspend collecting for that MIB object.

-n sourceName
Specifies the host names or IP addresses of the devices or nodes for which you want to collect data on the specified MIB object. The sourceName can indicate an individual node, or a set of nodes, based on a pattern-matching specification. This character string cannot contain spaces.

When you specify an IP address pattern-matching sourceName, the data collector will collect data on all nodes with an IP address in the specified
pattern-matching range. If the NetView program discovers a new node that has an address within the pattern-matching range, the data collector will automatically start data collection on the new node.

A valid IP address pattern-matching specification has the same syntax as a valid IP address with the addition that one or more of the dotted components of the address can be a range specified by an asterisk (*) or a more specific range specified by a dash (—). For example, the asterisk and dash can have the following meanings:

```
*.*.*.*
Collect on all possible addresses

15.2.112.*
Collect on all IP addresses on the subnet

15.2.112—120.*
Collect on all IP addresses on a range of subnets

15.2.112—120.177
Collect on the “177” machine(s) on a range of subnets
```

If you find there are certain nodes within the pattern-matching range on which you do not want to collect data, you can set up a configuration using the `collectionMode` option to exclude collection for those specific nodes to be excluded.

```
-c collectType
Specifies the collection type based on the values entered for `sourceName` and `collectionMode`. The following are valid values:

- `XW` The `sourceName` is a pattern-matching specification. It is excluded from collection (`collectionMode` value is `d`).
- `W` The `sourceName` is a pattern-matching specification. It is not excluded from collection.
- `XC` The `sourceName` is a specific node (not a pattern-matching specification). It is excluded from collection (`collectionMode` value is `d`).
- `C` The `sourceName` is a specific node (not a pattern-matching specification). It is not excluded from collection.
```

```
-m collectionMode
Specifies one of two valid values for the collection mode:

- `d` Excludes collection on a selected `sourceName`, even though it matches a pattern-matching specification. You can exclude either individual nodes or a range of nodes. A pattern-matching specification not to collect data takes precedence over an explicit specification to collect data.

  Alternately, specifying the `d` value does not store the MIB data, but does check thresholds, generating threshold events.

- `s` Stores the collected MIB data. If thresholds were previously set, it will continue to check them and generate threshold events. Otherwise, the MIB data is stored but thresholds are not checked.
```

```
-p pollingInterval
Determines how often data will be collected from the source. The format is:

  `<NUM> [smdhw]`
```
A positive real number followed by an s, m, h, or w indicates seconds, minutes, hours, or weeks, respectively. If you do not type a letter following the number, the default is seconds. For example, you can enter the following intervals:

- `1.5h` Indicates one and a half hours
- `30` Indicates 30 seconds

-v `thresholdVal`
Specifies a threshold value that indicates when you want to be notified of traffic patterns that are outside the usual expectations. When a collected value exceeds the specified threshold value, the data collector generates an event. This threshold value is required when you specify that a collection mode check thresholds. If the MIB object `SNMPDataType` is `Counter`, the threshold value is a float. If the MIB object `SNMPDataType` is `Gauge` or `Integer`, the threshold value is an integer.

-r `rearmVal`
Specifies a rearm value to control the frequency of threshold events generated. When a MIB value drops below or is equal to the specified rearm value, a rearm event is generated; another threshold event will not be generated until the rearm event occurs and the collected value again exceeds the specified threshold value after being rearmed.

If the `thresholdType` is set to `percent`, the rearm value should be a whole or fractional number in the range of 0–100. If the `thresholdType` is set to `absolute`, the rearm value should be a whole or fractional number and should be less than the `thresholdVal` value.

-t `thresholdType`
Specifies the threshold type, indicating the rearm value as either a percentage of the threshold value or as an absolute number. Specifying the rearm value as a percentage is useful, because if you change the threshold value, the rearm value will change proportionally. Type one of the following values:

- If you set the `collectionMode` to collect data and check thresholds, or to not collect data but to check thresholds, type either `%` for a percentage, or `A` for an absolute value.
- If you exclude collection or collect data without checking thresholds, type either `x%` for no percentage, or `xA` for no absolute value.

Note: When using the `x%` argument in a batch file, you must escape the `%` with another `%`. For example: `-tx%%`.

-T Specifies the NetView enterprise-specific trap to be sent when the threshold is exceeded. This threshold must be an odd value in the range of 1001 to 1999, or the default value of 58720263.

-a Specifies the threshold condition. Where `thresholdCondition` can be one of the following values: `=`, `!`, `>`, `>=`, `<`, `<=`, `CHANGED`, `NOTCHANGED`.

-D Use the `-D` flag to remove a threshold for MIB data Collection from the `\sys\ov\conf\snmpCol.conf` file. The `-i` and `-o` flags must also be specified to describe the instance number and the object ID name of the threshold to delete.

**Implementation Specifics**
The `thresh` command supports single-byte character code sets.
Files

\usr\lov\conf\snmpCol.conf
The snmpCollect daemon configuration file

Related Information
See \snmpCollect\ on page 680.
trapd

Purpose

Receives, logs, and forwards events and traps.

By default NetView tries to register with the SNMP Trap Service for traps, thus allowing other applications that register with the SNMP Trap Service to receive the same traps. If this fails because the SNMP Trap Service is not enabled or some other reason, trapd will attempt to listen on port 162 to the exclusion of other applications.

Syntax

```
trapd [-L]
trapd [-T]
trapd [-o]
trapd [-l logfile] [-n] [-t tracefile] [-x] [-b buffersize] [-a number] [-m number] [-z]
```

Description

The trapd daemon receives events and traps, logs them to the specified log file or database and forwards them to registering applications.

Events are messages that are sent by applications such as netmon. Traps are unsolicited messages sent by SNMP agents to SNMP network management stations. For example, the NetView SNMP agent sends traps to the NetView program.

The default log file, `/usr/ov/log/trapd.log`, is an ASCII file that grows without limit. Its entries are sorted by time, and each entry has the following format:

```
Seconds Number Time Node Source Description
```

These fields have the following meanings:

**Seconds**

The time the event or trap was received in seconds since the epoch (00:00:00 GMT January 1, 1970).

**Number**

A 2-digit number used for internal purposes.

**Time**

The time the event or trap was received.

**Node**

The node associated with the event or trap.

**Source**

A single character denoting where the event was generated. The following list contains defined sources:

- a: application
- A: Agent
- C: collect
- d: Demo
- D: Data Collector
- E: nvevents
- I: IPmap sa
You can define other source IDs through Options..Trap Setting.

**Description**
A description of the trap or event.

**Note:** Add or delete entries from the log file as needed, but do not modify the log file entries because they rely on the previously shown format.

The trapd daemon listens on UDP port 162 for SNMP traps from agents and a socket /usr/ov/sockets/trapd.socket for events from applications.

The trapd daemon can be configured through the NetView setup options. command or Options..Setup menu item.

**Flags**

- **-l logfile**
  Logs events and traps to the specified log file rather than the default file /usr/ov/log/trapd.log.

- **-n**
  Does not log events or traps.

- **-t tracefile**
  Turns on tracing and traces to tracefile, rather than the default trace file, /usr/ov/log/trapd.trace. Use the tracing facilities for debugging.

- **-x**
  Hexdumps all packets received by trapd and all packets sent from trapd. If tracing is turned off, this option will have no effect.

- **-L**
  Toggles trapd logging (see the -l option). This option is equivalent to sending a SIGUSR1 to the trapd that is currently running. If trapd is logging, it stops; if it is not logging, it starts. If trapd was started without logging or specifying logfile, it uses the default log file /usr/ov/log/trapd.log. The signal is sent to the process number found in /usr/ov/pids/trapd.pid.

- **-T**
  Toggles trapd tracing (see the -t option). This option is equivalent to sending a SIGUSR2 to the trapd that is currently running. If trapd is tracing, it stops; if it is not tracing, it starts. If trapd was started without tracing or specifying tracefile, it uses the default trace file, /usr/ov/log/trapd.trace. The signal is sent to the process number found in /usr/ov/pids/trapd.pid.

- **-o**
  Tells trapd to create a socket connection which allows NetView applications to register and receive SNMP trap information.

- **-b buffersize**
  Buffer size of the UDP receive buffer. This number must be between 9216 and 58254.

- **-a number**
  Tells trapd to purge events from the open database connectivity (ODBC) events.
database every time this number is reached. For example, -a 1250 purges the database every time there are 1250 records. The trapd daemon also purges the database on startup.

-m number
Specifies the maximum number of events to store in the ODBC database. When this number is reached, all excess records are purged. The trapd daemon also purges the database on startup.

-z  Connects to the ODBC database and stores events. The events are later displayed using the Event Browser.

Implementation Specifics
The trapd daemon supports single-byte character code sets.

You can configure and control trapd's options through nvsetup.

Files

\usr\io\log\trapd.log
Default log file

\usr\io\log\trapd.trace
Default trace file

\usr\io\sockets\trapd.socket
AIX Domain Socket

\usr\io\pids\trapd.pid
Process ID file

\usr\io\lrf\trapd_dm.lrf
LRF file

Related Information
- See "nvSnmpBlockingGetTable" on page 266.
trapd.conf

Purpose

This is the configuration file for trap formats.

Description

The file \usr\ov\conf\trapd.conf contains definitions for the handling of SNMP traps, including how to format trap log entries and what action to take, if any, when a trap is received. The trapd daemon uses these formats to log a message in the \usr\ov\log\trapd.log file. The formats are also used to present messages in the NetView Event Cards and List. Edit this file through the Options..Trap Setting operation.

The trapd.conf file has two types of entries: enterprise-definition and format-action-definition. An enterprise identifies the device or object generating the trap. All enterprise-definition entries are grouped at the beginning of the trapd.conf file. A format-action-definition specifies the logging format to use for a trap and what action to take.

Enterprise-Definition

The components of an enterprise-definition are separated by space characters.

Syntax:

ent-name ent-object-ID

Arguments are interpreted as follows:

ent-name
A character string containing no spaces, such as ibm.

ent-object-ID
The object ID number sequence containing no spaces enclosed in braces, such as \{1.3.6.1.4.1\}.

Format-Action-Definition

Each format-action-definition can contain an enterprise-identification, a format-specification, and an optional action-specification.

Enterprise-Identification:  The components of an enterprise-identification are separated by space characters.

Syntax:

trap-label ent-object-ID gen-trap spec-trap source-ID severity-level status-type category cmd-flag

Arguments are interpreted as follows:

trap-label
A short text label containing no spaces. This need not be unique and is used as a reading aid for the trapd.conf file.
**ent-object-ID**
The object ID number sequence containing no spaces enclosed in braces, such as \{1.3.6.1.4.1\}. This corresponds to the ent-object-ID from the enterprise-definition.

**gen-trap**
The generic-trap defined by SNMP as an integer from 0 (zero) to 6.

**spec-trap**
The specific-trap defined by SNMP as an integer value. This must be 0 (zero) if gen-trap is not 6.

**source-ID**
A single character denoting where the event was generated. The following list contains defined sources:

- a   application
- A   Agent
- C   collect
- d   Demo
- D   Data Collector
- E   nvevents
- I   IPmap sa
- L   loadmib
- M   IP topology
- n   netmon related
- N   netmon-generated traps
- O   OSI SA
- P   open topology trap (other than IP)
- S   Security Agent
- t   trap
- T   trapd
- V   vendor related

You can define other source IDs using the Options..Events..Trap Settings.

**severity-level**
A single character indicating the severity level for the trap.

- 0   Cleared
- 1   Indeterminate
- 2   Warning
- 3   Minor error
- 4   Critical
- 5   Major

**status-type**
A single character indicating the status that is to be assigned to an object when it generates the defined trap:
0  Defaults
1  Unknown
2  Up
3  Marginal
4  Down
5  Unmanaged
6  Acknowledge
7  User1
8  User2

category
A character string surrounded by double-quotation marks (""") indicating the category of the Events application in which to display the log message. Currently, the following strings are supported:
• “LOGONLY”
• “Threshold Events”
• “Network Topology Events”
• “Error Events”
• “Status Events”
• “Node Configuration Events”
• “Application Alert Events”

cmd-flag
A single character indicating the presence of an action-specification following the format-specification:
-  No action-specification
!  Action-specification present

Format-Specification: The format-specification consists of a single line of textual information to be logged to the Description column of `usr\ov\log\trapd.log` and in the Event Cards and List. You can enter several special characters to control the formatted output. The format can include standard C printf formats (for example, “\t” inserts a tab). To include information from the incoming trap, use the following formatting commands:

$C  Display the trap community string.
$E  Display the enterprise as a text string if possible, otherwise as in the $e argument.
$e  Display the enterprise as an Object ID string of numbers.
$A  Display the trap agent-addr using gethostbyaddr as a string. If this fails, use `inetoa` to display the agent-addr.
$G  Display the generic-trap.
$S  Display the specific-trap.
$T  Display the time-stamp. This is the remote machine’s time in hundredths of a second, between the last re-initialization of the network entity and the generation of the trap.
*S  Display all the variable-bindings supplied by the trap as name-type:value strings.
Display the number of variable-bindings in the trap.

Display the $ character.

Display the value of the nth variable-binding in the trap, where n is the variable-binding sequence number starting at 1 as it appears in VarBindList.

Display the nth variable-binding as a name-type:value string.

Display the nth variable-binding as a name:value string.

Display the nth variable-binding as a LANtern time string (INTEGER & Epoc=1/1/00).

Display corresponding text associated with the codepoint defined in the nth variable. Determine whether the next variable is associated with the subvector; if it is, display that text also. Continue until the next variable is not associated with the subvector.

Action-Specification: The action-specification consists of two lines. The first line is the command or program name to execute upon receipt of the incoming trap. The second line is the argument list passed to the command or program specified on the first line. This line is required even when the command takes no arguments.

User Events

Events are messages sent in the form of SNMP traps by applications such as the netmon daemon through the trapd daemon. They are used to inform applications that have registered with the trapd daemon that something of significance has happened. For example, the status of a node on the network might have changed or a threshold exceeded. Events are SNMP traps; their content should describe how they fit into the SNMP trap format. SNMP traps are composed of the following components:

enterprise
The object ID of the entity sending the trap. For events, this object ID will always be .1.3.6.1.4.1.11.2.3.2.

agent-addr
The IP address of the host sending the trap. This address will be 0.0.0.0 for events from applications on the local system.

generic-trap
A number identifying the type of trap. For events, this will be 6, which corresponds to enterpriseSpecific.

specific-trap
A number identifying the sub-type of trap.

time-stamp
The time since a well-known time in the past. For events, this time is 0.

variable-bindings
Extra data that provides information related to the trap. For events, the following conventions are followed:

<table>
<thead>
<tr>
<th>Nr</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An integer corresponding to the application sending the event. For the purposes of logging by the trapd daemon, this integer can be modified by appropriate entries in the /usr/ov/conf/trapd.conf file. Object ID: .1.3.6.1.4.1.11.2.15.2.0</td>
</tr>
</tbody>
</table>
Type: INTEGER

2 The node name for which the event applies or "<none>" for events not tied to a particular node.

Object ID: .1.3.2.1.4.1.11.2.15.3.0

Type: OCTET STRING

3 The description of the event. For the purposes of logging by the trapd daemon, this node name can be modified by appropriate entries in the \usr\ov\conf\trapd.conf file.

Object ID: .1.3.6.1.4.1.11.2.15.4.0

Type: OCTET STRING

4 Trap data specific to each event

Object ID: .1.3.6.1.4.1.11.2.15.5.0

Type: OCTET STRING

It is possible to simulate events occurring by using snmptrap, but, unless explicitly documented, this simulation is generally not supported.

Example: The following example shows the OV_SNMP_Set_Status event, which has a specific trap number of 58916871:

\usr\ov\bin\snmptrap 'hostname' \
   .1.3.6.1.4.1.11.2.3.2.3 "" 6 58916871 "" \
   .1.3.6.1.4.1.11.2.15.2.0 Integer 14 \
   .1.3.6.1.4.1.11.2.15.3.0 OctetString $OBJECT \
   .1.3.6.1.4.1.11.2.15.5.0 OctetString "Object status is" \
   .1.3.6.1.4.1.11.2.15.5.0 OctetString $NEWSTATUS

In this nvsnmptrap command, the agent-addr and time-stamp are defaulted using an empty string.

The value of the variable $OBJECT in the nvsnmptrap command must be the GUI object name. The object name for the map can be found by clicking on the symbol and selecting Object..Properties.

The value of the shell variable $NEWSTATUS can be set to one of the following values:

- Unknown
- Normal
- Marginal
- Critical

The value Up can be used for Normal, and Down for Critical. The object’s status will be set only when the ovw command is running.

If another application is responsible for controlling the status of an object, the last application to change the object’s status will be reflected in the map. For example, if the ipmap application changes an object’s status to Down, and an application changes the object’s status to Up using the nvsnmptrap command, the status of the object will be Up. For this reason, you should not change the status color of an object that the NetView program discovered and added.

To add an object that contributes to the status of a node, double click on the node symbol and add your object, such as an “Appli. Software” icon, with the object name $OBJECT. Ensure that the status source for $OBJECT is set to Object. You
can verify this by clicking on the symbol and selecting Describe/Modify Symbol.
Also, ensure that the status source for the node, the parent of $OBJECT, is set to
Compound. You can modify the propagation rules by changing the propagated
status through Edit..Properties.

Related Information

- See "trapd" on page 700.
- See "netview" on page 65.
- See "nvsnmptrap" on page 270.
- See "ovsnmp.conf" on page 304.
- See "ipmap" on page 37.
trapfrwd

Purpose

Forwards traps to another management station.

Syntax

```
trapfrwd
```

Description

The trapfrwd daemon forwards traps to other management stations.

The trapfrwd daemon uses the `\usr\OV\conf\trapfrwd.conf` file to determine where to send traps and what traps to send. This daemon is not started by default. To configure the trapfrwd daemon, modify the `trapfrwd.conf` file and then start the trapfrwd daemon.

The `trapfrwd.conf` file has the following format:

The first section contains the host name and trap port number pairs of the management stations to send traps. These pairs must be enclosed between the `[Hosts]` and `[End Hosts]` tags. For example:

```
[Hosts]
mgtsys1 1662
mgtsys2 0
[End Hosts]
```

The second sections contains the Enterprise OID and trap number pairs (the trap numbers are from the `trapd.conf` file). These pairs must be enclosed between the `[Traps]` and `[End Traps]` tags. For example:

```
[Traps]
1.3.6.1.4.1.2.6.3 591790
1.3.6.1.4.1.2.6.3 589824
[End Traps]
```

To add a comment, precede the line with a pound sign (`#`).

After you configure hosts and traps in the `trapfrwd.conf` file, start the trapfrwd daemon by entering the following commands on the command line:

```
ovaddobj \usr\ov\lrf\trapfrwd.lrf
ovstart trapfrwd
```

Files

`\usr\ov\conf\trapfrwd.conf`

Configuration file for the trapfrwd daemon.

`\usr\ov\lrf\trapfrwd.lrf`

Local registration file for the trapfrwd daemon.
webserver

Purpose

The webserver daemon is the Tivoli NetView Web server.

Syntax

webserver

Description

The webserver daemon runs the jetty server used by the Tivoli NetView Web consoles.

Dependencies

By default, the webserver daemon is not registered with ovspmd. It can be registered in either of two ways: using the `ovaddobj` command-line option, or through the Server Setup application.

The webserver daemon is dependent on other daemons (ovtopmd, trapd, owmdb) and, when registered with ovspmd, is started and stopped automatically with the other NetView daemons through the Server Setup application, or by using the `ovstart` and `ovstop` commands.

The `ovspmd` daemon uses the spmsur.exe application to launch the webserver daemon.

Related Information

- See `ovaddobj` on page 277
- See `ovdelobj` on page 279
- See `ovspmd` on page 354
- See `ovstatus` on page 360
- See `ovstart` on page 358
- See `ovstop` on page 361
- See `polld` on page 671
xxmap

Purpose

Provides the display of open topology information stored in the gtmd daemon.

Description

The xxmap application is integrated with the NetView program. The submap aggregation and content are defined by the protocol discovery applications or agents and are stored within the gtmd database. The xxmap application queries both the gtmd database and the ovwdb database for display and semantic information.

By default, the gtmd (and xxmd) daemon is not started when the NetView program is started. The gtmd daemon and the xxmd daemon are complementary. They must run in tandem.

Add the gtmd and xxmd daemons to the startup file by using the ovaddobj command. Add the -Restart flag to the /usr/OV/registration/C/xxmap file. Add the -Restart flag to the following line in this file:

`Command -Shared -Initial "${XXMapDir:-/usr/OV/bin}/xxmap";`

This line should read as follows:

`Command -Shared -Restart -Initial "${XXMapDir:-/usr/OV/bin}/xxmap";`

Adding the -Restart flag to the xxmap registration file enables automatic restarting of xxmap should this be necessary while the EUI is running. If you reconfigure the gtmd and xxmd daemons, so that they do not start automatically, you must manually remove the -Restart flag from the /usr/OV/registration/C/xxmap file.

The otenable.bat file adds, in the \usr\conf\ovsuf file, the gtmd and xxmd open topology daemons to be the list of daemons to start. After these daemons are added to the \usr\conf\ovsuf file, you can take advantage of the open topology features and NVOT APIs. The otdisable.bat file removes, from the \usr\conf\ovsuf file, the gtmd and xxmd open topology daemons from the list of daemons to start.

Map Synchronization: The xxmap application starts in a synchronization phase. While xxmap is synchronizing, the graphical interface displays the updating map message on the status line of the displayed submaps. During synchronization, xxmap brings the displayed maps up to date with the gtmd and ovwdb databases. During map synchronization, xxmap cannot respond to a Protocols request. The user may see the synchronizing message flash occasionally while gtmd is receiving topology information. This is expected behavior.

User Editing of Map: The xxmap application supports changes to the map through the graphical interface. You can add and delete symbols. However, added symbols are not verified or stored in the gtmd database. They will exist in the user plane. When you delete symbols from the map, the object represented by that symbol is deleted only when all symbols representing that object are deleted. That object, which is semantically stored in the gtmd database, is not deleted. The object can be deleted only by the agent that added that particular information. You can perform Cut and Paste operations on symbols, but the symbols will exist in the user plane after the paste operation. You must be careful when cutting and pasting
agent-added objects, because these operations are not reflected in the gtmd semantic database and can cause inconsistencies in the map.

**Root Submap:** The root submap is the basis of all protocols including IP. On the Root Submap window, a symbol is displayed for each protocol being managed. Each of these symbols will explode into a submap containing more symbols for that protocol. This submap can also contain symbols that explode into submaps. This aggregation can continue according to selections that you or the application make. Each protocol is represented in this tree structure as an individual branch starting at the root submap. The xxmap application allows you to jump between different protocol branches that contain the same node.

**oid_to_protocol data file:** The xxmap application uses the oid_to_protocol data file to convert from an oid provided in the topology MIB to a string that represents the graph protocol name. For more information about oid_to_protocol, see "oid_to_protocol" on page 272.

**Related Information**
- See "gtmd" on page 35.
- See "oid_to_protocol" on page 272.
Chapter 3. Using NetView GTM Data Structures

The data structures used by the general topology manager (GTM) API are defined in the nvoTypes.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 object database will be truncated to 256 characters. String fields in the GTM database can have unlimited length.

Basic Structures

These structures are used to represent the basic GTM entities:
- vertices
- graphs
- boxes
- arcs
- saps

These 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 {
    nvoOperationType operation;
    nvoAttributeBitmapType validAttributes;
    nvoVertexAttrType vertexAttr;
} nvoVertex;
typedef struct {
    int count;
    nvoVertex *vertex;
} nvoVertexList;

Graph

typedef struct {
    nvoOperationType operation;
    nvoAttributeBitmapType validAttributes;
    nvoGraphAttrType graphAttr;
} nvoGraph;
typedef struct {
    int count;
    nvoGraph *graph;
) nvoGraphList;
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;

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 {
    _nvotVertexType vertexType;
    _nvotVertexProtocolType vertexProtocol;
    char * vertexName;
    char * vertexMine;
    _nvotOctetString * vertexManagementExtension;
    _nvotOctetString * vertexManagementAddr;
    _nvotOperationalStateType vertexOperationalState;
    _nvotUnknownStatusType vertexUnknownStatus;
    _nvotAlarmStatusType vertexAlarmStatus;
} _nvotVertexAttrType;

_nvotGraphAttrType

typedef struct {
    _nvotGraphType graphType;
    _nvotGraphProtocolType graphProtocol;
    char * graphName;
} _nvotGraphAttrType;
The structures are the basic building blocks used to create basic structures and table structures.

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.
nvotGraphProtocolType

typedef char * nvotGraphProtocolType;

ctypedef union {
            nvotVertexProtocolType vertexProtocol;
            nvotGraphProtocolType graphProtocol;
        } nvotProtocolType;

716 Programmer’s Reference
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 reserved for future use 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 reserved for future use 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 reserved for future use SET_BIT( 9 )
#define ISROOT_ATTR SET_BIT( 10 )
#define reserved for future use 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 ADETAILINDEXID_ATTR SET_BIT( 4 )
#define ZDETAILINDEXID_ATTR SET_BIT( 5 )
#define reserved for future use SET_BIT( 6 )
#define ADETAILINDEXID_ATTR SET_BIT( 7 )
#define ZDETAILINDEXID_ATTR SET_BIT( 8 )
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 reserved for future use SET_BIT( 6 )

Sample

To use a nvotVertex structure where the attributes vertexProtocol and vertexName are filled, 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
} 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
} 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;

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;
Index

A
accessibility information xii
addtrap 20
appmon 25

B
basic structures, GTM 713

C
client/server functions, list of 15
commands, list of
SNMP 10

data structures, using GTM
basic structures 713
table structures 714
type structures 715
disability information xii

D
data structures, using GTM
basic structures 713
table structures 714
type structures 715
disability information xii

E
event 28
evtstat 30

F
findroute 31
format, reference page 19
functions, list of
IP topology 15
miscellaneous 15

G
graph 32
graphical user interface, list of
routines 1
GTM data structures, using.
basic structures 713
table structures 714
type structures 715
gtmd 35

I
information, accessibility xii
information, disability xii
IP topology functions, list of 15
ipmap 37

K
keyboard, shortcut keys xii

L
loadhosts 44
loadmib 46
lrf 48

M
mib2trap 57
mibExpr.conf 51
mibform 53
mibserver 55
mibtable 56
miscellaneous functions, list of 15

N
netmon 59
netview 65
NetView GTM data structures, using
basic structures 713
table structures 714
type structures 715
nmdemandpoll 68, 69
nmpolling 70
nvold 72, 73
nvecho 74
NVinstallMode 79
NVisClient 80
nvotChangeArcDetails 81
nvotChangeArcIconInGraph 85
nvotChangeArcLabelInGraph 89
nvotChangeArcStatus 94
nvotChangeBoxBackgroundColor 98
nvotChangeBoxDetails 101
nvotChangeBoxIconInGraph 104
nvotChangeBoxLabelInGraph 107
nvotChangeBoxPositionInGraph 110
nvotChangeGraphBackgroundColor 113
nvotChangeGraphDetails 116
nvotChangeGraphIconInGraph 119
nvotChangeGraphLabelInGraph 122
nvotChangeGraphPositionInGraph 125
nvotChangeRootGraphIcon 128
nvotChangeRootGraphLabel 131
nvotChangeVertexDetails 134
nvotChangeVertexIconInBox 137
nvotChangeVertexIconInGraph 140
nvotChangeVertexLabelInBox 143
nvotChangeVertexLabelInGraph 146
nvotChangeVertexPositionInBox 149
nvotChangeVertexPositionInGraph 152
nvotChangeVertexStatus 155
nvotCreateArcInGraph 158
nvotCreateBoxInGraph 164
nvotCreateGraphInGraph 169
nvotCreateProvidingSap 174
nvotCreateRootGraph 178
nvotCreateUsingSap 182
nvotCreateVertexInBox 185
nvotCreateVertexInGraph 189
nvotDeleteArc 193
nvotDeleteArcFromGraph 196
nvotDeleteBox 200
nvotDeleteBoxFromGraph 202
nvotDeleteGraph 205
nvotDeleteGraphFromGraph 207
nvotDeleteProvidingSap 210
nvotDeleteUsingSap 213
nvotDeleteVertex 216
nvotDeleteVertexFromBox 218
nvotDeleteVertexFromGraph 221
nvotDone 224
nvotFree 226
nvotFreeArc 226
nvotFreeArcList 226
nvotFreeBox 226
nvotFreeBoxList 226
nvotFreeGraph 226
nvotFreeGraphList 226
nvotFreeSap 226
nvotFreeSapList 226
nvotFreeVertex 226
nvotFreeVertexList 226
nvotGetArcsInGraph 228
nvotGetBoxesInGraph 231
nvotGetError 234
nvotGetErrorMsg 237
nvotGetGraphsInGraph 238
nvotGetSapsOnVertex 241
nvotGetVerticesInBox 244
nvotGetVerticesInGraph 247
nvotInit 250
nvotSetCenterBoxForGraph 253
nvotSetCenterGraphForGraph 256
nvotSetSynchronousCreation 259
npage 76
nvsniffer 263
nvSnmpBlockingGetTable 266
nvSnmpErrString 269
nvSnmpGetTable 266
nvSnmpGetTableElement 266
nvSnmptrap 270
nvSnmpWGetTable 266

O
oid_to_protocol 272
oid_to_sym 273
oid_to_type 275
ovaddobj 277
OVDefaultServerName 78
ovdelobj 279
OVFree 281
OVFreeSetupSelect 282
ovmapcount 283
ovmapdump 285
ovmapsnap 287
OVmib_get_objid_name 289
OVmib_read_objid 290
ovobjprint 291
OVsDone 356
OVSelect 293
OVSetPDUCommunityName 294
OVSetPDUIpAddress 295
OVSetupSelect 296
OVsInit 356
OVsInitComplete 356
ovsnp.conf 304
OVsnpAddNullVarBind 297
OVsnpAddTypedVarBind 297
OVsnpAddVarBind 297
OVsnpBlockingSend 299
OVsnpClose 302
OVsnpConfAllocEntry 307
OVsnpConfClose 308
OVsnpConfCopyEntry 309
OVsnpConfDeleteEntry 310
OVsnpConfFreeDest 311
OVsnpConfFreeEntry 312
OVsnpConfOpen 313
OVsnpConfReadDefault 316
OVsnpConfReadEntry 317
OVsnpConfResolveDest 319
OVsnpConfStoreDefault 321
OVsnpConfStoreEntry 323
OVsnpCreatePdu 325
OVsnpDoRetry 327
OVsnpErrString 328
OVsnpEventOpen 329
OVsnpFixPdu 331
OVsnpFreePdu 333
OVsnpGetRetryInfo 335
OVsnpIntro 337
OVsnpOpen 329, 342
OVsnpRead 345
OVsnpRecv 347
OVsnpSend 349
OVsnpTrapOpen 352
OVsnpWClose 302
OVsnpWOpen 342
OVsnpWSend 349
OVsnpWTrapOpen 352
ovsnpmd 354
OVsPMD_API 356
OVsReceive 356
ovstart 358
ovstatus 360
ovstop 361
ovtopmd 363
ovtopodump 366
ovtopofix 368
ovw_config 370
ovw_fields 371
ovw_verify 372
OVwAckMapClose 373
OVwAckUserSubmapCreate 375
OVwAddActionCallback 377
OVwAddAlertCallback 379
OVwAddCallback 382
OVwAddHelpCallback 384
OVwAddInput 386
OVwAddMenuItem 388
OVwAddMenuItemFunction 390
OVwAlertMsg 393
OVwApiIntro 395
OVwBeginMapSync 410
OVwCheckAction 412
OVwClearSymbolApp 628
OVwConfirmAddSymbolCB 652
OVwConfirmAppConfigCB 658
OVwConfirmCapabilityChangeCB 415
OVwConfirmCompoundStatusCB 433
OVwConfirmCreateObjectsCB 417
OVwConfirmCreateSubmapsCB 419
OVwConfirmCreateSymbolsCB 421
OVwConfirmDeleteObjectsCB 423
OVwConfirmDeleteSubmapsCB 425
OVwConfirmDeleteSymbolsCB 665
OVwConfirmDescribeCB 668
OVwConfirmHideSymbolsCB 427
OVwConfirmManageObjectsCB 429
OVwConfirmMoveSymbolCB 431
OVwConfirmObjectStatusCB 433
OVwConfirmSymbolStatusCB 433
OVwConfirmUnhideSymbolsCB 427
OVwConfirmUnmanageObjectsCB 429
OVwCopyMapInfo 544
OVwCountObjectsByFieldId 435
OVwCreateAction 436
OVwCreateApp 436
OVwCreateComponentSymbol 453
OVwCreateComponentSymbolByName 453
OVwCreateConnSymbol 453
OVwCreateConnSymbolByName 453
OVwCreateMenu 444
OVwCreateMenuItem 446
OVwCreateSubmap 449
OVwCreateSymbol 453
OVwCreateSymbolByHostname 453
OVwCreateSymbolByName 453
OVwCreateSymbolBySelectionName 453
OVwCreateSymbols 453
OVwDatabaseStillAlive 463
ovwdb 464
OVwDbAppendEnumConstants 466
OVwDbCreateField 468
OVwDbCreateObject 471
OVwDbCreateObjectByHostname 471
OVwDbCreateObjectBySelectionName 471
OVwDbDeleteField 468
OVwDbDeleteObject 473
OVwDbFieldIdToFieldName 475
OVwDbFieldNameToFieldId 475
OVwDbFreeEnumConstants 477
OVwDbFreeFieldBindList 486
OVwDbFreeFieldInfo 480
OVwDbFreeFieldList 496
OVwDbFreeFieldValue 482
OVwDbFindObjectFieldList 488
OVwDbFindObjectIdList 499
OVwDbGetCapabilityFieldValues 486
OVwDbGetEnumConstants 477
OVwDbGetEnumName 477
OVwDbGetFieldBooleanValue 482
OVwDbGetFieldEnumByName 482
OVwDbGetFieldEnumByValue 482
OVwDbGetFieldId 480
OVwDbGetFieldIntegerValue 482
OVwDbGetFieldStringValue 482
OVwDbGetFieldValues 486
OVwDbGetFieldValuesByObjects 488
OVwDbGetNameFields 486
OVwDbGetUniqObjectName 490
OVwDbHostnameToObjectld 492
OVwDbInit 494
OVwDbListFields 496
OVwDbListObjectsByFieldvalue 499
OVwDbListObjectsByFieldValues 499
OVwDbNameToObjectld 502
OVwDbObjectldToHostname 492
OVwDbObjectldToSelectionName 504
OVwDbSelectionNameToObjectld 504
OVwDbSetEnumConstants 506
OVwDbSetFieldBooleanValue 508
OVwDbSetFieldBooleanValue 508
OVwDbSetFieldEnumByName 508
OVwDbSetFieldEnumByValue 508
OVwDbSetFieldIntegerValue 508
OVwDbSetFieldStringValue 508
OVwDbSetFieldValue 508
OVwDbSetHostname 511
OVwDbSetSelectionName 511
OVwDbUnsetFieldvalue 513
OVwDbUnsetFieldValues 513
OVwDeleteAction 436
OVwDeleteApp 440
OVwDeleteMenu 444
OVwDeleteMenuItem 446
OVwDeleteSubmap 449
OVwDeleteSymbol 453
OVwDeleteSymbols 453
OVwDisplaySubmap 515
OVwDoAction 412
OVwDone 517
OVwEndMapSync 410
OVwEndSessionCB 518
OVwError 520
OVwErrorMsg 521
OVwEventIntro 522
OVwFileDescriptor 524
OVwFindMenuItem 525
OVwFreeActionRegInfo 436
OVwFreeAppRegInfo 440
OVwFreeMapInfo 544
OVwFreeObjectInfo 550
OVwFreeObjectList 569
OVwFreeSubmapInfo 556
OVwFreeSubmapList 571
OVwFreeSymbolInfo 558
OVwFreeSymbolList 574
OVwFreeSymbolTypeList 577
OVwGetAction 436
OVwGetApp 440
OVwGet AppConfigValues 527
OVwGetAppName 530
OVwGetClientList 531
OVwGetConnSymbol 532
OVwGetFirstAction 535
OVwGetContextMenu 537
OVwGetFirstMenuItemFunction 540
OVwGetFirstRegContext 542
OVwGetMappingInfo 544
OVwGetMenuItem 446
OVwGetMenuItemMenu 546
OVwGetMenuItemPath 546
OVwGetNextAction 535
OVwGetNextMenuitem 537
OVwGetNextMenuItemFunction 540
OVwGetNextRegContext 542
OVwGetObjectInfo 550
OVwGetRegContext 552
OVwGetSelections 554
OVwGetSubmapInfo 556
OVwGetSymbolInfo 558
OVwGetSymbolsForObject 560
OVwHighlightObject 562
OVwHighlightObjects 562
OVwInit 564
OVwInitApiEvents 566
OVwIsIdEqual 567
OVwIsIdNull 567
OVwListObjectsOnMap 569
OVwListSubmaps 571
OVwListSymbols 574
OVwListSymbolTypeCaps 577
OVwListSymbolTypes 577
OVwLockRegUpdates 579
OVwMainLoop 581
OVwMapCloseCB 583
OVwMapOpenCB 585
OVwPeekInputEvent 587
OVwPeekOVwEvent 587
OVwPending 589
OVwProcessEvent 590
OVwQueryAddSymbolCB 652
OVwQueryAppConfigCB 658
OVwQueryConnectSymbolCB 661
OVwQueryDeleteSymbolCB 665
OVwQueryDescribeCB 668
OVwRegIntro 591
OVwRemoveActionCallback 377
OVwRemoveAlertCallback 379
OVwRemoveCallback 382
OVwRemoveHelpCallback 384
OVwRemoveInput 386
OVwRemoveMenuitem 388
OVwRemoveMenuitemFunction 390
OVwRenameRegContext 615
OVwSaveRegUpdates 617
OVwSelectListRegContext 619
OVwSetAction 436
OVwSetApp 440
OVwSetAppConfigValues 527
OVwSetBackgroundGraphic 621
OVwSetMenuitem 446
OVwSetMenuPathSeparator 548
OVwSetRegContext 552
OVwSetStatusOnObject 623
OVwSetStatusOnObjects 623
OVwSetStatusOnSymbol 623
OVwSetStatusOnSymbols 623
OVwSetSubmapName 626
OVwSetSymbolApp 628
OVwSetSymbolBehavior 630
OVwSetSymbolLabel 633
OVwSetSymbolPosition 635
OVwSetSymbolStatusSource 640
OVwSetSymbolType 643
OVwShowHelp 646
OVwSubmapCloseCB 648
OVwSubmapOpenCB 649
OVwUndoRegUpdates 617
OVwUnlockRegUpdates 579
OVwUserSubmapCreateCB 650
OVwVerifyAdd 652
OVwVerifyAppConfigChange 658
OVwVerifyConnect 661
OVwVerifyDeleteSymbol 665
OVwVerifyDescribeChange 668

P
polld 671

Q
QuickTest 672

R
reference page format 19
metstat 673
routines, list of
graphical user interface 1
SNMP 10

S
schema for NetView databases 675
shortcut keys, keyboard xii
smartsetmaint 677
smartsetutil 678
SNMP, list of
commands 10
routines 10
snmpCollect 680
snmpget 683
snmpnext 685
snmpodump 687
snmpset 690
snmptrap 270
snmpwalk 692
spmsur.exe 694

thresh 696
topology functions, list of IP 15
trapd 700, 710
trapd.conf 703
type structures, GTM 715

X
xxmap 711

table structures, GTM 714
TESS 695