

HP OpenView Network Node Manager Administrator's Reference

HP 9000 Series



**Manual Part Number: J2316-90005
Printed in U.S.A., 12/93**

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, and specifically disclaims the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for any damages, whether direct, indirect, consequential, incidental, or special, in connection with the furnishing, performance, or use of this material.

The specific warranty information for the product as to which this material relates is available in Hewlett-Packard's purchasing and licensing terms and conditions.

© Copyright 1992, 1993 Hewlett-Packard Company.

All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

© Copyright 1980, 1984, AT&T, Inc.

© Copyright 1988, 1989, Carnegie Mellon University.

© Copyright 1988, 1989, Massachusetts Institute of Technology.

© Copyright 1979, 1980, 1983, 1985-1988 The Regents of the University of California.

© Copyright 1986, 1987, 1988 Sun Microsystems, Inc.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

OSF/Motif is a trademark of the Open Software Foundation.

UNIX® is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

Hewlett-Packard Company
3404 East Harmony Road
Fort Collins, CO 80525 U.S.A.

© Copyright 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678,

Legend: Rights Reserved

001 73TA 4291 001 1000000000

© Copyright 1988 1989 Carnegie Mellon University

Copyright 1988, 1989 Massachusetts Institute of Technology.

1980-1981, 1982-1983, 1984-1985, 1986-1987, 1988-1989, 1990-1991, 1992-1993, 1994-1995, 1996-1997, 1998-1999, 2000-2001, 2002-2003, 2004-2005, 2006-2007, 2008-2009, 2010-2011, 2012-2013, 2014-2015, 2016-2017, 2018-2019, 2020-2021, 2022-2023, 2024-2025, 2026-2027, 2028-2029, 2030-2031, 2032-2033, 2034-2035, 2036-2037, 2038-2039, 2040-2041, 2042-2043, 2044-2045, 2046-2047, 2048-2049, 2050-2051, 2052-2053, 2054-2055, 2056-2057, 2058-2059, 2060-2061, 2062-2063, 2064-2065, 2066-2067, 2068-2069, 2070-2071, 2072-2073, 2074-2075, 2076-2077, 2078-2079, 2080-2081, 2082-2083, 2084-2085, 2086-2087, 2088-2089, 2090-2091, 2092-2093, 2094-2095, 2096-2097, 2098-2099, 2100-2101, 2102-2103, 2104-2105, 2106-2107, 2108-2109, 2110-2111, 2112-2113, 2114-2115, 2116-2117, 2118-2119, 2120-2121, 2122-2123, 2124-2125, 2126-2127, 2128-2129, 2130-2131, 2132-2133, 2134-2135, 2136-2137, 2138-2139, 2140-2141, 2142-2143, 2144-2145, 2146-2147, 2148-2149, 2150-2151, 2152-2153, 2154-2155, 2156-2157, 2158-2159, 2160-2161, 2162-2163, 2164-2165, 2166-2167, 2168-2169, 2170-2171, 2172-2173, 2174-2175, 2176-2177, 2178-2179, 2180-2181, 2182-2183, 2184-2185, 2186-2187, 2188-2189, 2190-2191, 2192-2193, 2194-2195, 2196-2197, 2198-2199, 2200-2201, 2202-2203, 2204-2205, 2206-2207, 2208-2209, 2210-2211, 2212-2213, 2214-2215, 2216-2217, 2218-2219, 2220-2221, 2222-2223, 2224-2225, 2226-2227, 2228-2229, 2230-2231, 2232-2233, 2234-2235, 2236-2237, 2238-2239, 2240-2241, 2242-2243, 2244-2245, 2246-2247, 2248-2249, 2250-2251, 2252-2253, 2254-2255, 2256-2257, 2258-2259, 2260-2261, 2262-2263, 2264-2265, 2266-2267, 2268-2269, 2270-2271, 2272-2273, 2274-2275, 2276-2277, 2278-2279, 2280-2281, 2282-2283, 2284-2285, 2286-2287, 2288-2289, 2290-2291, 2292-2293, 2294-2295, 2296-2297, 2298-2299, 2300-2301, 2302-2303, 2304-2305, 2306-2307, 2308-2309, 2310-2311, 2312-2313, 2314-2315, 2316-2317, 2318-2319, 2320-2321, 2322-2323, 2324-2325, 2326-2327, 2328-2329, 2330-2331, 2332-2333, 2334-2335, 2336-2337, 2338-2339, 2340-2341, 2342-2343, 2344-2345, 2346-2347, 2348-2349, 2350-2351, 2352-2353, 2354-2355, 2356-2357, 2358-2359, 2360-2361, 2362-2363, 2364-2365, 2366-2367, 2368-2369, 2370-2371, 2372-2373, 2374-2375, 2376-2377, 2378-2379, 2380-2381, 2382-2383, 2384-2385, 2386-2387, 2388-2389, 2390-2391, 2392-2393, 2394-2395, 2396-2397, 2398-2399, 2400-2401, 2402-2403, 2404-2405, 2406-2407, 2408-2409, 2410-2411, 2412-2413, 2414-2415, 2416-2417, 2418-2419, 2420-2421, 2422-2423, 2424-2425, 2426-2427, 2428-2429, 2430-2431, 2432-2433, 2434-2435, 2436-2437, 2438-2439, 2440-2441, 2442-2443, 2444-2445, 2446-2447, 2448-2449, 2450-2451, 2452-2453, 2454-2455, 2456-2457, 2458-2459, 2460-2461, 2462-2463, 2464-2465, 2466-2467, 2468-2469, 2470-2471, 2472-2473, 2474-2475, 2476-2477, 2478-2479, 2480-2481, 2482-2483, 2484-2485, 2486-2487, 2488-2489, 2490-2491, 2492-2493, 2494-2495, 2496-2497, 2498-2499, 2500-2501, 2502-2503, 2504-2505, 2506-2507, 2508-2509, 2510-2511, 2512-2513, 2514-2515, 2516-2517, 2518-2519, 2520-2521, 2522-2523, 2524-2525, 2526-2527, 2528-2529, 2530-2531, 2532-2533, 2534-2535, 2536-2537, 2538-2539, 2540-2541, 2542-2543, 2544-2545, 2546-2547, 2548-2549, 2550-2551, 2552-2553, 2554-2555, 2556-2557, 2558-2559, 2560-2561, 2562-2563, 2564-2565, 2566-2567, 2568-2569, 2570-2571, 2572-2573, 2574-2575, 2576-2577, 2578-2579, 2580-2581, 2582-2583, 2584-2585, 2586-2587, 2588-2589, 2590-2591, 2592-2593, 2594-2595, 2596-2597, 2598-2599, 2600-2601, 2602-2603, 2604-2605, 2606-2607, 2608-2609, 2610-2611, 2612-2613, 2614-2615, 2616-2617, 2618-2619, 2620-2621, 2622-2623, 2624-2625, 2626-2627, 2628-2629, 2630-2631, 2632-2633, 2634-2635, 2636-2637, 2638-2639, 2640-2641, 2642-2643, 2644-2645, 2646-2647, 2648-2649, 2650-2651, 2652-2653, 2654-2655, 2656-2657, 2658-2659, 2660-2661, 2662-2663, 2664-2665, 2666-2667, 2668-2669, 2670-2671, 2672-2673, 2674-2675, 2676-2677, 2678-2679, 2680-2681, 2682-2683, 2684-2685, 2686-2687, 2688-2689, 2690-2691, 2692-2693, 2694-2695, 2696-2697, 2698-2699, 2700-2701, 2702-2703, 2704-2705, 2706-2707, 2708-2709, 2710-2711, 2712-2713, 2714-2715, 2716-2717, 2718-2719, 2720-2721, 2722-2723, 27

100-443887-200 5-21-64

OSF-Mont is a trademark of the Open Source Foundation.

UNIVY is a registered trademark of UNIVY Systems Laboratory, Inc. in the U.S.A. and other countries.

For Collins CO 80525 1 254
1404 East Harrison, 1030
Hewlett-Packard Company

Contents

Chapter 1: Introduction and Operational Overview

Documentation Guide and References	1-2
Online Help	1-2
HP-UX System Administration Manuals	1-2
Networking Manuals	1-2
HP OpenView Manuals	1-2
SNMP Concepts	1-3
TCP/IP Concepts	1-3
Internet Request for Comments Documents	1-4
Definitions and Concepts	1-5
Manager and Agents	1-5
MIBs	1-6
Internet Objects	1-6
How MIBs are Organized	1-7
Unique Properties of the SNMP MIB Object sysObjectID	1-8
HP OpenView Network Node Manager	1-11
SNMP Processes and Files	1-12
Start-Up Behavior of SNMP Processes and Files	1-12
SNMP Background Processes	1-14
Operational Behavior of SNMP Processes and Files	1-14
Background Processes	1-15
Foreground Processes	1-18
Databases	1-22

Chapter 2: HP OpenView Operations

Removing Products Installed with ovinstall	2-2
Starting the Manager Software on the Manager System	2-3
Running HP OpenView Windows without Network Polling	2-6
Stopping the Manager Software	2-7
Exiting from the Graphical Network Map	2-7
Stopping Background Processes	2-7
Restarting Automatic Map Generation	2-8

Chapter 3: Configuration

Modifying the Start-Up Configuration	3-2
The Local Registration File	3-2
Contents of the Local Registration File	3-2
Local Registration File Descriptions	3-3
Configuring netmon	3-5
Seed File Format	3-6
Configuring ovwdb	3-7
Configuring the SNMP Management Station	3-10
Proxy Configuration	3-10
Community Configuration	3-10
Set Community Configuration	3-11
Timeout and Retry Configuration	3-11
Remote Port Configuration	3-11
netmon Configuration	3-11
Configuring the Mapping from sysObjectID to Types	3-12
Editing the oid_to_sym Configuration File	3-12
Editing the oid_to_type Configuration File	3-14
Adding Values for vendor and SNMP Agent Fields	3-17
Configuring Terminal Connectivity to Remote HP 3000s	3-18
Integrating Existing Applications	3-20
Application Integration with HP OpenView Windows	3-20
Creating an Application Registration File	3-20
Modifying an Existing Application	3-23
Automatic Start-Up of Your Application	3-23
Changing File or Map Permissions	3-25
Defining New Symbol Classes	3-26
On-Going System Maintenance	3-26
File Clean-Up	3-26
snmpCollect/	3-26
ovsuf	3-27
Setting Up Automatic Map Snapshots	3-27
Removing Old Snapshots	3-28
Backing Up HP OpenView	3-29
Restoring HP OpenView From a Backup	3-30

Chapter 4: Troubleshooting

Recommended Practices	4-2
Logging and Tracing	4-3
Using Command Options	4-4
Using trapd, netmon, ovactiond, and snmpCollect	4-4
Using nettl and netfmt	4-6
Capturing Logging and Tracing Output with nettl	4-8
Characterizing the Problem	4-17
Scope: What is Affected?	4-17
Distinguish Local vs. Remote Problem	4-17
Remote Nodes' Level of Manageability	4-17
Affected Parts of the HP OpenView Network Management System	4-17
Context: What Changed?	4-17
Duration: How Long or How Often?	4-17
Context: What Action Was Performed?	4-18
General Product Troubleshooting Considerations	4-19
When You Need More Information	4-19
Troubleshooting Activation Key Problems	4-20
Troubleshooting IP Discovery and Layout	4-20
How IP Discovery and Layout Work	4-20
Why IP Discovery May Not Discover All Nodes	4-21
What to Do if IP Discovery Fails	4-22
General Suggestions	4-22
Common Symptoms	4-24
Troubleshooting Background Processes	4-28
The ovspmd Process	4-30
Determining ovspmd Problems	4-30
Recreating the ovspmd Sockets Directory	4-31
Recreating the ovsuf Configuration File	4-32
The netmon Background Process	4-33
The ovtopmd Background Process	4-35
The ovwdb Background Process	4-38
The trapd Background Process	4-40

Troubleshooting Operations	4-44
Runtime Components	4-45
Manager File Permissions	4-45
Start-Up Scripts	4-46
X Window Components	4-47
Network Management Operations	4-48
Execution Models for Network Management Operations	4-50
Browsing Internet MIBs	4-51
Building and Executing MIB Applications	4-52
Building MIB Applications	4-52
Executing MIB Applications	4-52
Troubleshooting Data Collector Problems	4-53
Configuring Events	4-56
Loading Internet MIBs	4-57
Known Limitations	4-57
Online Help	4-59
Performance and Traffic	4-60
Performance	4-60
Traffic	4-61
Appendix A: Upgrading from NNM 2.x to NNM 3.x	
Backing up NNM 2.x	A-1
File Placements and Formats	A-2
Files Requiring Manual Conversion	A-2
netmon Threshold Configuration	A-2
“Drop In” Applications	A-4
oid_to_type File	A-5
Files Automatically Converted	A-5
Menu Structure	A-7
Appendix B: Creating and Using Registration Files	
Appendix Overview	B-1
Application Registration Files	B-2
Application Block	B-3
Version and Application Description	B-4
Command	B-5
HP OpenView Windows Shell Environment Variables	B-5

Process Flags	B-7
Namefield	B-8
Menu Registration	B-9
Defining the Menu Structure	B-9
Invoking the Application	B-11
Symbol Type Registration Files	B-14
Defining Icon Symbol Classes	B-15
Defining Classes Using Line Segments.	B-15
Defining Classes Using Arcs	B-18
Optional Class Specifications	B-19
Defining Symbol Subclasses	B-22
Required Subclass Specifications	B-22
Optional Subclass Specifications	B-23
Field Registration File	B-24
Defining Fields	B-24
Field Type	B-25
Field Flags	B-25
Example Field Definitions	B-27
Combinations of Flags	B-28
How Registration Files Are Processed	B-29
Summary	B-30

Appendix C: Reference

Index

Introduction and Operational Overview

This manual is for the person who administers and troubleshoots the HP OpenView Network Node Manager.

This manual assumes that you are a knowledgeable HP-UX system and network administrator and troubleshooter. You should know how to do the following:

- Update your system with new software.
- Kill processes.
- Write scripts.
- Modify X Windows resources.
- View, search, and edit files.

This manual covers the topics of configuration and troubleshooting for the HP OpenView Network Node Manager. Also included is reference information for the product.

Information on installing or updating your HP OpenView software is found in the *Installation Guide*.

Information on how to use the graphical user interface of the HP OpenView Network Node Manager is available through an online **help** operation and the the HP OpenView Network Node Manager User's Guide. The focus of this manual is on information you need either when the HP OpenView Network Node Manager product is not running or when you need to understand the product's internals.

Documentation Guide and References

Online Help

- The graphical user interface, HP OpenView Windows, has comprehensive online help entries to help you understand and use the product. For example, through the **Help** menu, you can get help information on how to perform a particular task (**Help: Index -> Tasks**) or help information on a particular function or capability of the product (**Help: Index -> Functions**). In addition, each dialog box has a help button that provides access to information specific to the dialog box you are in.
- The online man pages.

Note

Refer also to the release note files in the `/usr/OV/ReleaseNotes` directory for information available too late to be documented elsewhere.

HP-UX System Administration Manuals

- *HP-UX Reference*

Networking Manuals

- *Installing and Administering LAN/9000*
- *Administering ARPA Services*
- *Using ARPA Services*

HP OpenView Manuals

- *Installation Guide:*
 - HP OpenView Network Node Manager*
 - HP OpenView SNMP Management Platform/Developer's Kit*
 - HP OpenView Distributed Management Platform/Developer's Kit*

- 
- *HP OpenView SNMP Platform Performance and Configuration Guide, with HP OpenView Network Node Manager Examples*
 - *HP OpenView Windows User's Guide*
 - *HP OpenView Network Node Manager User's Guide*
 - *HP OpenView SNMP Agent Administrator's Reference*
 - *HP OpenView Windows Application Design and Style Guide*
 - *HP OpenView Windows Programmer's Guide*
 - *HP OpenView Windows Programmer's Reference*
 - *HP OpenView SNMP Programmer's Guide and Reference*
 - *HP OpenView Integration Utilities Guide*
 - *Creating Product Packages for HP-UX*

SNMP Concepts

Rose, Marshall T. *The Simple Book: An Introduction to Management of TCP/IP-based Internets*. Englewood Cliffs, New Jersey: Prentice-Hall, 1989. To order, reference ISBN 0-13-812611-9.



TCP/IP Concepts

Comer, Douglas. *Internetworking With TCP/IP: Principles, Protocols, and Architecture*. Englewood Cliffs, New Jersey: Prentice-Hall, 1988. To order, reference ISBN 0-13-470154-2.

Internet Request for Comments Documents

Copies of these documents are shipped on the HP OpenView Network Node Manager product installation media. They are in the **ovSNMP-RUN** fileset and are installed in the **/usr/ov/doc** directory.

To find out about...	See...	Filename
how MIB objects are defined	RFC1155: Structure and Identification of Management Information for TCP/IP-based Internets	rfc1155.txt
the protocol used to manage the MIB objects	RFC1157: A Simple Network Management Protocol (SNMP)	rfc1157.txt
the format for creating MIB object files	RFC1212: Concise MIB Definitions	rfc1212.txt
the objects contained in the Internet-standard MIB	RFC1213: Management Information Base for Network Management of TCP/IP-based Internets: MIB-II	rfc1213.txt



Definitions and Concepts

To understand how the HP OpenView Network Node Manager manages a network, you need to become familiar with some concepts and terms. This section describes the concepts of the network management model and the management information base (MIB).

Manager and Agents

A manageable network consists of one or more manager systems, or network management stations, and a collection of agent systems, or network elements.

- A **manager system** executes network management operations which monitor and control agent systems. The implementation of these network management operations is called the **manager**.
- An **agent** is the interface to a managed object. An **agent system** is a device, such as a host, gateway, terminal server, hub, or bridge, that has an agent responsible for performing network management operations requested by the manager.



A manager and agent may exist on the same system.

The Simple Network Management Protocol (SNMP) communicates management information between a manager and an agent. SNMP permits the following activities:

- A manager can retrieve (**get**) management information from an agent. The manager sends requests for information to the agent, and the agent sends back replies containing the information requested.
- A manager can retrieve the name and value of the next instance in the managed object using the **get-next** operation.
- A manager can alter (**set**) management information on an agent.
- An agent can send information to the manager without an explicit request from the manager. Such an operation in SNMP is called a **trap**.

Traps proactively alert the manager of changes that occur on the agent system, such as a reboot. The agent knows which manager system to send traps to through a configurable trap destination.

SNMP requests for information on an agent are accompanied by a **community name**, which enables the manager to access that information. The manager accesses the agent's MIB object instances using SNMP's *get*, *getNext*, and *set* operations. Figure 1-1 shows a simplified diagram of the manager-agent interactions just described.

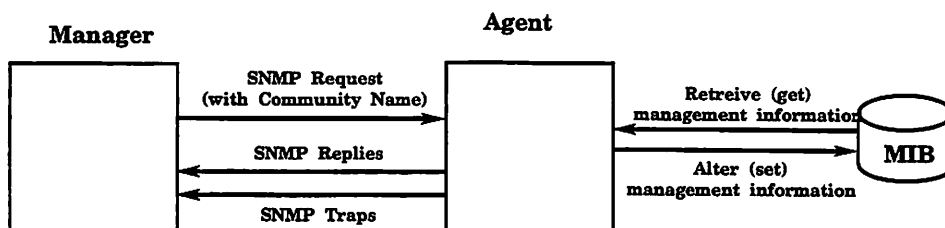


Figure 1-1. Manager-Agent Communication through SNMP

MIBs

Conceptually, the information on the agent is known as the **Management Information Base (MIB)**. The MIB is not a physically distinct database, but rather logically encompasses configuration and status values normally available on the agent system. A specific type or class of management information is called a **MIB object** (for example, a system description or an interface status). The existence of a particular value for a MIB object in the agent database is called an **instance**. Some MIB objects have only a single instance for a given agent system (for example, system description). Other MIB objects have multiple instances for a given agent system (for example, interface status for each interface on the system).

Internet Objects

Internet MIB objects are defined using the Internet-standard structure of management information (SMI) and compose a virtual data store on the agent system. This structure is defined by *RFC 1155: Structure and Identification of Management Information for TCP/IP-based Internets* and amended by *RFC 1212: Concise MIB Definitions*. RFCs 1155 and 1212 define the structure of management information for SNMP-based management. SNMP agents contain the “intelligence” required to access MIB values.

Internet MIBs are organized into MIB modules. A **MIB module** is a file defining all the MIB objects under a subtree. The foundation module is the standards-based MIB-II module defined by *RFC 1213: Management Information Base of Network Management of TCP/IP internets: MIB-II*. In addition to the Internet-standard MIB-II objects defined in RFC 1213, many hardware vendors, such as Hewlett-Packard, Cisco Systems, Wellfleet, and Novell, have developed MIB extensions for their own products. The MIBs defined by these various hardware vendors are referred to as **enterprise-specific MIBs** in this manual.

Many additional standard MIBs are under development (for example, host resources, bridges, hubs, and routers). These emerging standard MIBs are based upon a collaboration between vendors, academia, and

1-6 Definitions and Concepts

standards organizations. The intent is to consolidate common management information from enterprise MIBs into standard definitions, yielding consistent access to the management information across multivendor equipment.

How MIBs are Organized

Conceptually, the **MIB objects** are organized in a hierarchical tree structure. Each branch in the tree has a unique name and numeric identifier (see Figure 1-2). Intermediate branches of the tree serve as a way to group related MIB objects together. The “leaves” of the tree represent the actual MIB objects. A subtree is used to refer to the entire group of branches and leaves under a particular intermediate branch. Figure 1-2 illustrates the tree structure, with special attention to the Internet MIB tree.

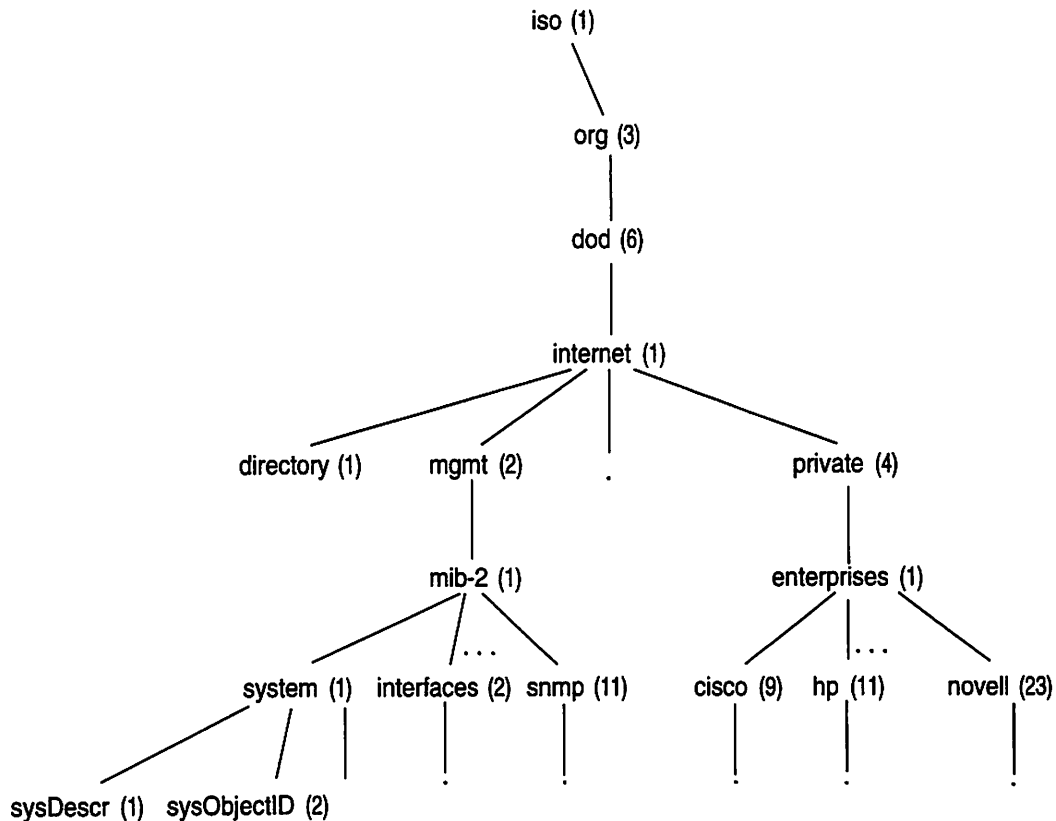


Figure 1-2. MIB Organization

A MIB object is uniquely identified (named) by its place in the tree. A full object identifier consists of the identifier of each branch along the path through the tree hierarchy, from the top of the tree (for example, **iso**) down to the leaf (for example, **sysObjectID**). The object identifier is conventionally

expressed in “dot notation,” by separating each branch identifier along the path with a period. For example, the **mib-2** subtree is **iso.org.dod.internet.mgmt.mib-2** and its numeric identifier is **1.3.6.1.2.1**. As another example, the full MIB object identifier for **sysObjectID** is **iso.org.dod.internet.mgmt.mib-2.system.sysObjectID** and its numeric identifier is **1.3.6.1.2.1.1.2**. These MIB object notations follow the standard notation defined in Abstract Syntax Notation One (ASN.1); the ASN.1 standard notation definition can be considered the “template” for MIBs.

For an SNMP MIB object, the last part of an object identifier is called the instance identifier. The instance identifier on an SNMP MIB object with only one instance is zero. The instance identifier on an SNMP MIB object with more than one instance is one or greater.

To avoid conflicts of object IDs, each branch of the tree must be registered with a designated organization. For example, the Internet Activities Board (IAB) has authority over the **internet** subtree, which includes the MIB-II Internet standard registered under the **mib-2** subtree. In turn, the IAB gives vendors authority over enterprise-specific subtrees. Enterprise-specific MIB objects are registered under the designated authority for that enterprise. To clarify this point, HP registers its enterprise-specific MIBs under **1.3.6.1.4.1.11**, having authority over the **enterprises.hp** subtree; and, Cisco registers its enterprise-specific MIBs under **1.3.6.1.4.1.9**, having authority over the **enterprises.cisco** subtree.

Unique Properties of the SNMP MIB Object **sysObjectID**

The **sysObjectID** is an important SNMP MIB object to HP OpenView network management operations. This section describes

- what the **sysObjectID** is
- how HP OpenView network management operations use the **sysObjectID**

The **sysObjectID** is registered in the Internet-standard MIB-II module as **iso.org.dod.internet.mgmt.mib-2.system.sysObjectID** (**1.3.6.1.2.1.1.2**). The **sysObjectID** is used for administrative purposes to uniquely identify the type of agent software that is running on a given vendor's hardware. This object is different from most other MIB objects. When you query it, you get back an object identifier that describes the product. For example, when you query the **iso.org.dod.internet.mgmt.mib-2.system.sysObjectID** using the MIB Browser on an HP OpenView SNMP Agent running on an HP 9000 Series 300, SNMP returns the object identifier

iso.org.dod.internet.private.enterprises.hp.nm.system.hpux.hp9000s300

This result tells you how the HP 9000 Series 300 agent is registered in the MIB. Its numeric identifier is **1.3.6.1.4.1.11.2.3.2.2**.

Note that there is no value associated with the object ID returned in **sysObjectID**. It is a unique identifier.

The **sysObjectID** includes both hardware and software information and will vary based on the type of hardware and agent software. Some vendors have a different **sysObjectID** for every version of their agents. For example, Hewlett-Packard sells several SNMP agents, each uniquely identified. To distinguish the HP 9000 Series 800 agent from the HP 9000 Series 300 agent, HP registered the HP 9000 Series 800 agent as **iso.org.dod.internet.private.enterprises.hp.nm.system.hpux.hp9000s800**.

netmon, a background process that automatically discovers the nodes on your network, obtains the **sysObjectID** when it queries SNMP nodes. The HP OpenView network management operations take advantage of this special object in the following ways:

- To describe the vendor; that is, the manufacturer of the hardware.
- To describe the source of the SNMP agent, which is the type of agent software running on the hardware. For example, if you do a GetRequest to a Sun SPARCstation, the vendor is Sun, but the SNMP agent may be
 - an HP OpenView SNMP Agent for SunOS
 - an SCO UNIX for SunOS agent
 - other SNMP agent software
- To determine IP topology behavior. An agent's **sysObjectID** provides hints about what type of node the agent is, for example, a gateway, a bridge, or a host.
- To determine symbol type when putting symbols for newly discovered nodes on the map, for example, a bridge, a gateway, or a hub.
- To identify the source of an event. A **sysObjectID** is sent in every trap. You use the **sysObjectID** when you configure events. The reference in the **Event Configuration** dialog box to enterprise ID is generally synonymous with the **sysObjectID** of the agent that generated the trap.

Example Use of sysObjectID

Here are some examples of how understanding the vendor and SNMP agent capabilities will benefit you when you use the HP OpenView network management operations.

- You can use the **Locate: Objects -> By Attribute** menu item to locate objects on the map by selecting the “vendor” or “SNMPAgent” attribute.
- You can determine the value of the **sysObjectID** associated with a node by selecting the **Monitor: Description -> Selected Objects** menu item and view the “IP Map” attributes of the selected object.

- Menu items are greyed-out or made available based on “vendor” or “SNMPAgent” attributes.

For example, the **Monitor: System Activity -> Disk Space...** menu item displays information that is obtained from the HP enterprise-specific MIB. This information is provided only by HP OpenView SNMP Agents. The **Disk Space** menu item, when registered in HP OpenView Windows (OVW), is registered such that the menu item is only active when you select a node that has the “SNMPAgent” attribute set to “HP 9000/HP-UX.” If you select a Cisco node, the **Disk Space** menu item will be greyed-out because its “SNMPAgent” attributes will not match the attributes for which the **Disk Space** menu item is registered.

For more information, see the following:

- The section in this chapter on “Processes and Files” describes the relationship between the different processes and files mentioned above.
- Chapter 3 describes how to configure the network management system to map the **sysObjectID** to OVW symbol types and to vendor, SNMP agent, and IP topology behaviors.
- Chapter 4 has information on how to troubleshoot problems caused by incorrect **sysObjectID** mappings. For example, why an object might show up as a generic computer in the OVW map.



HP OpenView Network Node Manager

The HP OpenView Network Node Manager (NNM) product is a configuration, performance, and fault management application for multivendor TCP/IP (Transmission Control Protocol/Internet Protocol) networks. The HP OpenView Network Node Manager lets you:

- Automatically discover the devices on the TCP/IP network and monitor the status of those devices.
- Automatically draw the Internet Protocol (IP) topology maps based on discovered information. A map is a graphical and hierarchical representation of your network and its systems. Discovered devices are placed in appropriate segments, networks, or internets based on the topology of the IP network.
- Manage any vendor devices that support the Simple Management Protocol (SNMP). NNM can manage standard MIB objects, as well as, enterprise-specific MIB objects.
- Include new enterprise-specific Internet MIBs into the HP OpenView network management system's Management Information Base (MIB). Once you have loaded the new MIB module on the management station, you can manage any of the MIB objects defined in that MIB module.
- Get and set values of Internet MIB objects using a point-and-click **Browse MIB** operation.
- Build new MIB applications, without programming, for Internet-standard and enterprise-specific MIBs. Once you have built your MIB application, you can monitor the objects through the HP OpenView Windows menu bar.
- Collect historical MIB information about the MIB objects, store MIB data for trend analysis, and graph collected data.
- Define event thresholds for MIB objects, for example, when the disk usage on a particular device exceeds a limit.
- Define actions to be taken upon receipt of an SNMP (Simple Network Management Protocol) trap.
- Integrate your own applications with the HP OpenView Windows menu bar to yield an integrated management solution.
- Diagnose network faults and performance problems from one location, and diagnose problems by looking at trends over time. This includes customizing and automating the monitoring of your network and the management station's response to events.

SNMP Processes and Files

This section defines the processes and files that are key to the operation of the SNMP Management Platform component of the HP OpenView Network Node Manager product. Included are explanations of the interactions and relationships among these processes and files. The interactions among processes and files at start-up time and during regular operation are different.

Start-Up Behavior of SNMP Processes and Files

Figure 1-3 shows the interactions and relationships among SNMP processes and files at start-up time.

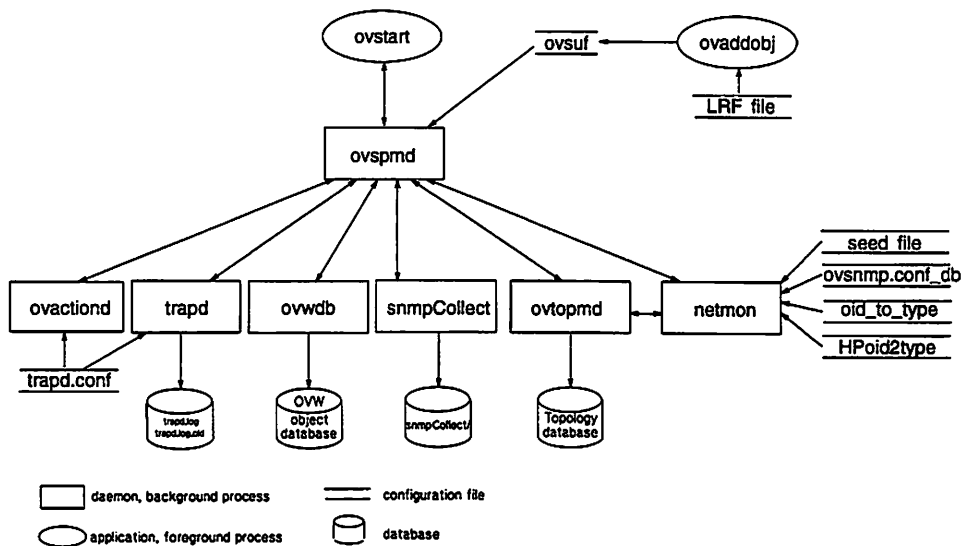


Figure 1-3. SNMP Processes and Files at Start-UP

ovstart

The program used to start the various processes (daemons) that make up the HP OpenView Network Node Manager product. When you execute the **ovstart** command, **ovstart** first contacts the process management daemon, **ovspmd**, to see if it is running. If **ovspmd** is not running, **ovstart** starts **ovspmd**. Once **ovspmd** is running, **ovstart** sends commands to **ovspmd** to start the background processes according to the start-up configuration file **ovsuf**. The **ovstart** command is automatically executed when you use the default installation. When you reboot the system, the **/etc/netnmrc** script automatically executes **ovstart**.

ovspmd

Maintains a communication channel for requests from **ovstart** and sends status responses to **ovstart**. It starts the processes **trapd**, **ovwdb**, **ovactiond**, **snmpCollect**, **ovtopmd**, and **netmon**. Each process is a child of **ovspmd** and must run under the control of **ovspmd**. The **ovspmd** process also maintains a communication channel with each process it has started. **ovspmd** uses this channel to send commands to the processes and to receive status responses from them.

/usr/OV/conf/ovsuf

Contains the configuration information for **ovspmd**. Each entry in **ovsuf** is created by **ovaddobj** from information in the LRF (Local Registration File). There is at least one LRF for each of the background processes. Some background processes may have more than one LRF. The LRF is used to tell **ovstart** what processes to start, what its dependencies are, and what the command line arguments are. **ovspmd** determines which processes to start first (based on the dependencies listed in the LRFs) and starts the processes in the correct order. You cannot directly edit the **ovsuf** file. You can modify **ovsuf** using **ovaddobj** and **ovdelobj**.

SNMP Background Processes

This section lists the background processes that should always be running. You can check the status of these processes by executing the **ovstatus** command. Note that in addition to reporting the status of the background processes, **ovstatus** reports the status of **ovspmd**. For more details about these processes, see the next section “Operational Behavior of SNMP Processes and Files.” All of the following executables reside in **/usr/ov/bin**:

netmon	The background process that polls SNMP agents to initially “discover” your network topology and then detect topology, configuration, and status changes in the network.
ovtopmd	The background process that maintains the network topology database.
ovwdb	The background process that controls the OVW object database.
trapd	The background process that multiplexes and logs SNMP traps. trapd logs all SNMP traps to the files /usr/ov/log/trapd.log and /usr/ov/log/trapd.log.old .
ovactiond	The background process that executes commands upon receipt of an event.
snmpCollect	The background process that collects MIB data and performs threshold monitoring. snmpCollect stores the data it collects in the /usr/ov/databases/snmpCollect directory and sends all threshold events to trapd .

Operational Behavior of SNMP Processes and Files

Figures 1-4A and 1-4B show the operational interactions and relationships among SNMP processes and files.

There are two types of processes

- Background processes or daemons that run continuously independent of whether **ovw** is running. The background processes are shown as rectangles in figures 1-4A and 1-4B.
- Foreground processes that run while **ovw** is running. The foreground processes are shown as ovals in figures 1-4A and 1-4B.

Background Processes

This section describes the SNMP background processes that are running during regular operation. All of the following files reside in **usr/OV/bin**:

- netmon** Polls SNMP agents to initially “discover” your network topology and then detect topology, configuration, and status changes in the network. You may optionally provide **netmon** with a “seed file” containing a list of nodes in your administrative domain to define the initial management region of the automatically-generated network map. Without the seed file, **netmon** assumes that your initial administrative domain is composed of the networks to which the management station is directly connected. You may expand this initial management region interactively using the graphical network map operations **Manage Objects** and **Unmanage Objects** under the **Options** menu. **netmon** uses the **oid_to_type** and **HPoid2type** files to determine correct IP topology behavior and correct vendor and SNMP agent values.
- ovtopmd** Maintains the network topology database. The topology database is a set of files in the **/usr/OV/databases/openview/topo** directory that stores **netmon** polling values and information about network objects, including their relationships and status. **ovtopmd** reads the topology database at start-up. If the topology database does not initially exist, **ovtopmd** creates it, and **netmon** populates it with network objects during discovery of your network's topology and automatic generation of the map. Once it has been generated, **ovtopmd** controls the topology database and updates specific records based on information from **netmon** indicating that changes have occurred in the corresponding nodes.
- ovwdb** Controls the **OVW object database**. The OVW object database stores information about the objects that HP OpenView Windows uses for **Describe** operations, (for example, when you select the **Describe/Modify Object** operation). These objects may be represented on the map by one or more symbols. If **netmon** detects a change in the network, **netmon** calls **ovtopmd** which calls **ovwdb** to update the OVW object database. **ovwdb** must be running for HP OpenView Windows to work.

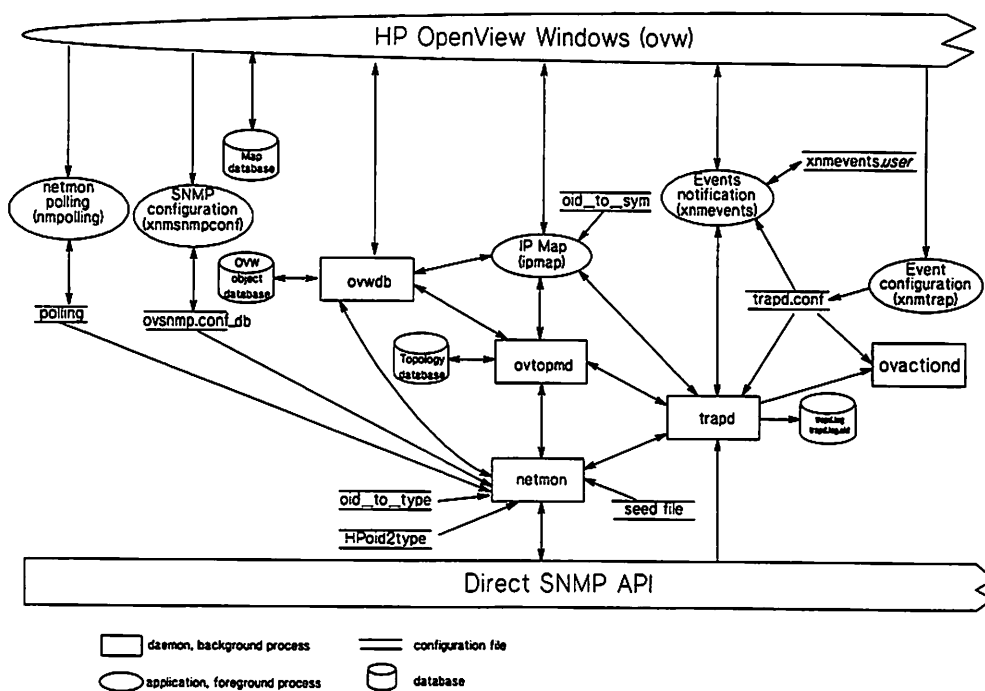


Figure 1-4A. Processes and Files During Operation (Part 1)

- trapd** Multiplexes and logs SNMP traps. **trapd** logs all SNMP traps to the files `/usr/ov/log/trapd.log` and `/usr/ov/log/trapd.log.old`. This ASCII file serves as both the network event log, and the log file for **trapd** and **netmon** errors. **trapd** also forwards SNMP traps from the network on to other applications that have connected to **trapd** using the SNMP API.
- ovactiond** Receives events from **trapd** and executes commands.
- snmpCollect** The background process that collects MIB data and performs threshold monitoring. **snmpCollect** stores the data it collects in the `/usr/ov/databases/snmpCollect` directory and sends threshold events to **trapd**.

Foreground Processes

This section describes the SNMP foreground processes that are running when you execute **ovw**. All of these files reside in **/usr/ov/bin**:

ovw The HP OpenView Windows process that provides map drawing, map editing, and menu management. When you start **ovw** by executing the **ovw** command, the **ovw** process automatically invokes **ipmap** and **xmmevents**. The Map database is used to store presentation information about the maps, such as symbol location or symbol types, for objects in the OVW object database. A map is a set of related sub-maps that provides a graphical and hierarchical presentation of your network and its systems. **ovwdb** must be running for **ovw** to work.

You can take a snapshot of the Map database. A snapshot is an image of the map database that you can reference later for planning purposes. A snapshot appears as a static picture of the map at a specific time. Refer to chapter 3 for more information on snapshots.

Once the HP OpenView network management system is operational, the **ipmap**, **ovtopmd**, **netmon**, **snmpCollect**, **ovactiond**, and **xmmevents** processes are dynamically updated through events forwarded by the **/usr/ov/bin/trapd** process. **trapd** logs incoming events to the **/usr/ov/log/trapd.log** file and forwards them to **ipmap** for reflection in the graphical network map, to **xmmevents** for display (as pending events) in the **Events Browser** window, and to **netmon** for use in tracking the state of the network.

ipmap

Runs under **ovw** to automatically draw IP topology maps representing your IP network in the HP OpenView Windows graphical user interface. The map of your IP network is referred to as IP Map in the HP OpenView manuals.

When **ovw** starts **ipmap**, **ipmap** first queries **ovw** to find out when the IP Map was last open and then calls **ovtopmd** to find out everything that has changed since the last time the IP Map was open. This start-up process is called the synchronization phase. This phase indicates that the map may not be completely up-to-date. As changes come in, **ipmap** updates IP Map to reflect the changes and saves the changes in the Map database. If **ipmap** can match a **sysObjectID** to a symbol type in the **oid_to_sym** file, the node will be shown with an appropriate symbol in IP Map. If the **oid_to_sym** file does not have a matching **sysObjectID** entry, **ipmap** will use the topology attributes. For example, if the topology attribute is set to gateway, but there is no symbol for it in the **oid_to_sym** file, it will be shown as a generic gateway. Nodes that do not talk SNMP will show up as generic computers.

Once the synchronization phase is completed, IP Map is updated based on information received dynamically from **netmon**, which continuously monitors the state of your network using SNMP requests and ICMP requests to remote systems. **netmon** communicates any changes it finds to **ovtopmd** and sends corresponding events to **trapd**. **ovtopmd** updates the topology database and the OVW object database. If the change affects the presentation, **ipmap** will act on it; otherwise the change is only reflected in the OVW object database.

Likewise, when you edit the map, change **netmon**'s polling values, or when the status of nodes propagates up to the segment or network containing them, **ipmap** updates the topology database with the changes and sends corresponding events to **trapd**. Once the changes have been added to the topology database, **ovtopmd** notifies **ipmap**, and the map of your IP network is automatically updated.

xnmevents Automatically invoked by **ovw** to display pending SNMP traps. **xnmevents** reads the **/usr/OV/log/xnmevents.username.map** file only at start-up to recover SNMP traps that have not previously been deleted and its last exit date and time. With its last exit date and time, **xnmevents** determines which SNMP traps in **/usr/OV/log/trapd.log** happened since the last time **xnmevents** was running. **xnmevents** reads the **/usr/OV/log/trapd.log** file only at start-up to obtain those pending SNMP traps. **xnmevents** also reads **trapd.conf** to obtain information about how to customize trap messages. Following start-up, **xnmevents** receives SNMP traps forwarded from **trapd**, and updates the **Events Browser** window. Upon exit, the **xnmevents** process writes all SNMP traps that have not been deleted to the **xnmevents.username.map** file, along with the date and time of exit. Note: If you are using a short file name system, be sure the first 14 characters of the file name are unique.

The OVW application **xnmevents** process checks every five minutes to see if **netmon**, **snmpCollect**, are running. If any process is not running, **xnmevents** will display a dialog box and report the inactive process. (For information on changing the five-minute default polling interval or on disabling this polling feature, see the **/usr/OV/app-defaults/XNm** file.)

xnmtrap The process invoked by the **Options: Event Configuration -> SNMP...** menu item that enables you to handle enterprise-specific events (traps). You can customize the message displayed through **xnmevents** when a particular event arrives. You can also specify a command or a script that should be executed when a particular event arrives. Event configuration changes are stored in the **/usr/OV/conf/c/trapd.conf** configuration file.

xnmsnmpconf This is the **Options: SNMP Configuration...** menu item that enables you to add, delete, and modify SNMP configuration parameters and the **netmon** status polling interval. The SNMP configuration parameters include community name, set community name, timeout interval, number of retries, and proxy information. Manager processes such as **netmon**, **snmpCollect**, **xnmbrowser**, **xnmgraph**, MIB applications, and SNMP API-based applications use these parameters when they access agent MIB values through SNMP requests.

xnmloadmib This is the **Options: Load/Unload MIBs -> SNMP...** menu item used to load new Internet-standard or enterprise-specific MIBs into the Loaded MIB database. Once you have loaded the new MIB into the Loaded MIB database, you can use the Browse MIB, MIB Data Collection, MIB Application Builder, and the applications built by the MIB Application Builder to manage your multivendor network.

xnmbrowser This is the **Monitor: MIB Values -> Browse MIB...** menu item that enables you to get and set MIB values for Internet-standard and enterprise-specific MIB objects.

xnmcollect The **Options: Data Collection & Thresholds -> SNMP...** menu item that enables you to configure the manager to collect MIB data from network objects at regular intervals. The configuration information is stored in the `/usr/OV/conf/snmpCol.conf` configuration file. The collected data are stored in files in the `/usr/OV/databases/snmpCollect` directory. The file names will have the format of *label.instance*, therefore, if you are using a short file name system, be sure the first 14 characters of the file name are unique. Once you have stored the collected data, you can use **xnmgraph** to view the collected data or import the data into your own application using **snmpColDump**. You can also define thresholds for the collected MIB data and generate events when the specified thresholds are exceeded.

xnmbuilder The **Options: Application Builder -> SNMP...** menu item that enables you to build custom screens to manage multivendor MIB objects. The information you define using the MIB Application Builder is stored in Registration files and Help files. **xnmgraph** is used to display “Graph” applications; while **xnmappmon** is used to display “Form” and “Tables” applications.

xnmgraph The HP OpenView Graph tool that enables you to graph the results of monitoring operations for managed SNMP objects selected from the map. The results may be real-time or collected historical data. When an operation is selected from the HP OpenView Windows menu bar, **ovw** forwards the objects selected on the map to **xnmgraph**. **xnmgraph** retrieves the data (if any) from the Data Collector database, queries the selected objects and displays the results in a line graph.

xnmappmon The HP OpenView Application Encapsulator that displays textual results of monitoring operations for managed SNMP objects selected from the map. When an operation is selected from the OVW menu bar, **ovw** forwards the objects selected on the map to **xnmappmon** as input. **xnmappmon** puts up the appropriate dialog box, translates the selected operation and objects into a command and executes it. Upon completion, the command returns its text output to **xnmappmon** for display in the dialog box. In the case of **Diagnose: Network Connectivity -> Locate Route**, **xnmappmon** also returns to **ovw** a list of map objects to highlight as output of the network management operation.

nmpolling This is the **Options: Topology/Status Polling** menu item that updates the polling configuration. **nmpolling** also monitors the following background processes:

netmon
ovtopmd
snmpCollect

nmpolling monitors these processes to determine if one of them has gone down and displays a dialog box if a process is down.

mibform,
mibtable,
rnetstat,
findroute,
rdbf, and
rpig Commands executed by **xnmappmon** that enable you to monitor and diagnose problems in your TCP/IP network.

Databases

This section describes the differences between the Map database, the OVW object database, and the topology database. These databases are controlled by the HP OpenView processes; you cannot edit them directly.

The Map database. The Map database contains primarily presentation information. This presentation information is specific to each map. There is one Map database per map. Examples of presentation information that is stored in the Map database include the exact symbol placement on the map, the symbol associated with each object, and symbol labels. **ovw** updates the Map database based on requests from the user or other processes such as **ipmap**.

The OVW object database. The OVW object database contains global object information. The information is generic; that is, it is not customized to any specific application. The OVW object database contains information related to fields such as **sysObjectID**, vendor, and SNMP agent. You can see the fields in the OVW object database through the **Describe** dialog box. If **netmon** detects a status change in the network, **netmon** calls **ovtopmd** to update the OVW object database. There is only one object database per system.

The Topology database. The topology database contains topology information used by IP discovery and layout (**netmon**, **ovtopmd**, **ipmap**). The information in the topology database spans all maps. Much of the information in the OVW object database is duplicated in the topology database. Information in the topology database that is not in the OVW object database includes state information for **netmon**. The most important **netmon** state information includes time stamps indicating when the object last changed and should next be polled.

This information helps **netmon** detect changes so **netmon** can communicate the changes to **ovtopmd** and to **trapd**. The topology database is controlled by **ovtopmd** and is updated based on information received from **netmon**. There is only one topology database per system.

HP OpenView Operations

This chapter includes the following information:

- instructions for removing products installed with **ovinstall**
- instructions for starting the manager software and running HP OpenView Windows without polling
- instructions for stopping the manager software, exiting from the HP OpenView Windows graphical network map, and stopping background processes
- instructions for restarting automatic map generation

Note

For instructions on installing the HP OpenView Network Node Manager, see the *Installation Guide*.

Removing Products Installed with ovinstall

Products that are installed using **ovinstall** can be removed using **ovremove**. To do so, follow these steps.

1. Remove the HP OpenView Network Node Manager product by typing

```
/usr/OV/bin/ovremove -p NNMGR
```

2. Remove the HP OpenView SNMP Management Platform product by typing

```
/usr/OV/bin/ovremove -p SNMPRUN
```

See the **ovremove** man page for more details.

Starting the Manager Software on the Manager System

Once you start the HP OpenView Network Node Manager software, you can expect intense polling traffic at first as the **netmon** process works to discover what is on your network as quickly as possible. The new node polling frequency automatically decreases as **netmon** finds fewer new objects per polling cycle; other polling traffic (such as for status monitoring) remains constant.

Only IP objects are discoverable by **netmon**. Initially, the IP map will contain only

- IP networks and gateway/routers on the Internet submap
- bus segments, gateway/routers, and hubs/bridges on the Network submap
- hosts, gateway/routers, and hubs/bridges on the Segment submap

Some IP nodes that are not computers (but rather connectors, such as bridges and repeaters, or devices) appear on the map as hosts. However, Hewlett-Packard connectors that support IP and SNMP appear on the map as the appropriate connector. The HP OpenView network management system, when putting symbols on the map, matches the **sysObjectID** to a symbol type to be used in the map. If the system can match a **sysObjectID** to a symbol type, the node will be shown with the appropriate symbol in the map; if not, the node will be represented as a generic symbol. For more information about **sysObjectID**, see chapter 1 of this manual.

A node appears as a gateway if its MIB value for IP forwarding is set to "forwarding" and the node has at least two interfaces. IP nodes with multiple interfaces on the same network may appear multiple times on segments. A node may also appear as a gateway if the gateway flag is set in the **oid_to_type** file. For more information, see "Editing the **oid_to_type** Configuration File" in chapter 3 of this manual.

After the map is generated, you can refine it through editing. See the *HP OpenView Network Node Manager User's Guide*.

To start the HP OpenView Network Node Manager software on the manager system, perform the following steps.

1. If you have not already done so, set your path to include `/usr/OV/bin`.

```
PATH=$PATH:/usr/OV/bin      Use with /bin/ksh or /bin/sh:
export PATH
```

```
set path=($path /usr/OV/bin) Use with /bin/csh:
```

2. If you have purchased and want to use the SNMP agent product, execute the `/etc/chksnmpd` command to verify that the SNMP agent process is running. `netmon` depends on the agent running.

If you run `chksnmpd` as root, and the agent software is running correctly, the results look similar to the following:

```
chksnmpd: hostname, Fri Apr 26 12:43:06 1993
        SU privileges
        Reading file      : /etc/snmpd.conf
        Sending request :
Success: snmpd responded to request
```

3. You must be root to perform this step. To verify that the background processes are running, execute the `/usr/OV/bin/ovstatus` command. If the background processes are not running, execute the `/usr/OV/bin/ovstart` command. The `ovstart` command starts the background processes.
4. If you are not running X Windows or HP VUE, start it.
5. If you want to redirect your X Windows display to a system other than the management system
 - a. Set your X Windows `DISPLAY` variable on the HP OpenView network management system using one of the commands below. Replace `hostname` with the host name of the system to which you are redirecting the display.

```
DISPLAY=hostname:0.0      Use with /bin/ksh or /bin/sh
export DISPLAY
```

```
setenv DISPLAY hostname:0.0 Use with /bin/csh.
```

2-4 Starting the Manager Software on the Manager System

- b. Make sure that the management system has permission to display windows on **hostname**. If the management system does not have permission, use the **xhost** command to add the management system to the **xhost** table on the **hostname** system. To do so, on **hostname** type

xhost + managementsystem

where **managementsystem** is the host name of the management system.

6. Start the graphical network map (user interface) by typing

ovw

ovw is executable by anyone.

Alternatively, you can run **ovw** in the background to free up the terminal window for other uses. In some cases, **ovw** prints error messages to standard output and standard error. To capture these messages and to prevent jobs from stopping, you may want to redirect messages to a temporary file. To run **ovw** in the background and to redirect error messages to a temporary file, type

ovw > /tmp/ovw.log 2>&1 &

Once the graphical network map is up, explore it! An online **Help** operation is ready to assist your experimentation, explain map appearances or product behavior when they are not what you expected, and serve as reference when you forget the details of an operation. If you are wondering how to expand and customize your map, investigate the **Options** menu's **Manage Objects** and **Unmanage Objects** operations and the **Edit** menu's operations.

Running HP OpenView Windows without Network Polling

You can run HP OpenView Windows, **ovw**, without **netmon**, the background process that polls your network to generate the map and then detect changes. This may be useful if the only applications you have running under HP OpenView Windows do not rely on **netmon**'s discovery functionality.

To run **ovw** without automatic IP discovery, configuration checking, and polling by **netmon**, follow these steps:

1. Stop the **netmon** process by typing

```
ovstop netmon
```

2. Modify the start-up configuration file by typing

```
ovdelobj /usr/OV/lrf/netmon.lrf
```

If you choose to restart **netmon**, do the following:

3. Verify that the **trapd**, **ovwdb**, and **ovtopmd** background processes are running by typing

```
ovstatus
```

You should see a list of which processes are running. If any of the above three processes are not running, refer to the appropriate section in chapter 4 describing that process ("ovwdb Background Process", "trapd Background Process", or "ovtopmd Background Process".) Then follow the instructions given to restart the problem process.

1. Modify the start-up configuration file by typing

```
ovaddobj /usr/OV/lrf/netmon.lrf
```

2. Restart the **netmon** process by typing

```
ovstart netmon
```

Stopping the Manager Software

If you want the HP OpenView network management system to continuously monitor and track changes to your network and the management system, always keep **trapd**, **ovwdb**, **ovtopmd**, and **netmon** running, even if the graphical network map (**ovw**) is not up. If you want to continuously collect data, you also need to keep **snmpCollect** running. If you want to perform event actions, keep **ovactiond** running. These processes are automatically started when you install the HP OpenView product.

To stop the HP OpenView network management system, exit the graphical network map and optionally stop the manager's background processes using the **ovstop** command.

Exiting from the Graphical Network Map

To exit from the graphical network map (**ovw**), select **Exit** from the **File** menu.

Stopping Background Processes

Refer to the "Troubleshooting Background Processes" section in chapter 4 for information on background processes that depend on other processes to operate properly.

To stop the background processes, use the **ovstop** command. You must be root to run the **ovstop** command.

To stop all the processes, type

```
ovstop
```

To stop an individual process, for example, the **netmon** process, type

```
ovstop netmon
```

To stop several processes at the same time, list each process that you want to stop. For example, to stop **ovwdb**, **ovtopmd**, and **netmon**, type

```
ovstop ovwdb ovtopmd netmon
```

Restarting Automatic Map Generation

If a particular network on the map does not look right, use the graphical user interface to edit the map. However, if the entire map is misrepresented and does not accurately represent your network, you may want to restart automatic map generation. To restart automatic map generation, you need to recreate your map and topology databases. Examples of when you may want to restart automatic map generation include:

- If nodes are discovered slowly, then start **netmon** with a seed file.
- The IP topology of your entire management domain has changed dramatically.
- The map or topology databases are corrupted (for example, have run out of disk space).

To recreate your map and topology databases, as root, perform the following steps.

1. Exit the HP OpenView Windows graphical user interface by selecting the **File: Exit** menu item.
2. Execute the **ovstop** command. The **ovstop** command stops all the background processes.
3. Remove the map databases. Note that this step destroys all existing maps—you may wish to back up your existing databases. Then execute the following command:

```
rm -rf /usr/OV/databases/openview/*/*
```

4. Remove the **/usr/OV/log/xmmevents.username.map**, **/usr/OV/log/trapd.log**, and **/usr/OV/log/trapd.log.old** files.
5. If you are using the **Ingres** option, execute the **ovstart ovdbmon** command as superuser and **ovtoposql -C** as user **ingres**. For more information see the *HP OpenView Data Administrator's Reference* manual.
6. Restart the background processes to generate a new map by typing the following commands:

```
/usr/OV/bin/ovstart ovwdb  
/usr/OV/bin/ovw -fields  
/usr/OV/bin/ovstart
```

If you have used the **Options: Topology/Status Polling: IP** menu item to configure **netmon**, you will need to reconfigure those operations again.

You are now ready to restart the HP OpenView Windows graphical user interface. See the section “Starting the Manager Software on the Manager System” earlier in this chapter.

Configuration

This chapter discusses ways you may optionally configure the HP OpenView Network Node Manager. The chapter explains how to do the following:

- Modify the start-up configuration.
- Modify **netmon**'s configuration for controlling IP discovery.
- Configure terminal connectivity to remote HP 3000s.
- Integrate your own applications into the HP OpenView Windows menu bar and modify an application's Application Registration File.
- Change file and map permissions.
- Define new symbol classes.
- Optimize the management system's performance through on-going system maintenance.

Modifying the Start-Up Configuration

The start-up process, **ovspmd**, obtains information about which process to start from the start-up file, **ovsuf**. The **ovsuf** file is created by the **ovaddobj** command, which adds start-up information based on Local Registration Files (LRFs). For more information about the start-up process, see chapter 1.

To modify the start-up configuration, you edit the LRF to change the command line options. This section contains the following information:

- A definition of the LRF and a general description of the LRFs used by the HP OpenView network management system.
- The procedure for how to configure the HP OpenView network management system to start **netmon** with a seed file. See “Configuring netmon.”
- The procedure for how to configure the HP OpenView network management system to reduce the amount of memory used by the **ovwdb** process. See “Configuring ovwdb.”

The Local Registration File

The Local Registration File (LRF) is a specially formatted ASCII file that contains information about a process such as its name, where its executable code is located, how to start it, and the objects, if any, for which it is responsible.

Contents of the Local Registration File

The LRF consists of at least two lines and, optionally, comments. Here is an example of the first two lines in the **ovtopmd.lrf** file.

```
#
# OpenView ovtopmd LRF
#
ovtopmd:/usr/OV/bin/ovtopmd:
Ovs_YES_START:trapd,ovwdb:-O:Ovs_WELL_BEHAVED:15:
```

Each line in the LRF contains fields separated by colons.

The number symbol (#) indicates the beginning of a comment, which continues to the end of the line.

To modify the start-up configuration, change the command line options. The command line options are specified in the third field of the second line. In the example `ovtopmd.lrf` above, the third field of the second line shows the `-o` option.

For a step-by-step procedure on how to edit the LRFs, follow the same steps as for editing the `netmon.lrf` file. See the following section “Configuring netmon.”

Local Registration File Descriptions

Each background process (except `ovspmd`) has an associated LRF. The HP OpenView Network Node Manager LRFs are located in the `/usr/OV/lrf` directory.

Table 3-1 describes each LRF, and any options applied as defaults in the LRF. The table does not contain a complete list of options. To learn about all the options available for each LRF, see the man page for the appropriate process.

Table 3-1. Local Registration Files (LRFs)

LRF	Description
netmon.lrf	The LRF for netmon, the background process that polls agents to initially “discover” your network. Default option: -P which means that it is started through ovstart. You may want to use the following options: -b <burstcount> Add if you have an unreliable network. This allows you to do multiple pings on a node. -s <seedfile> Add when you want the initial map to include specific networks in your administrative domain. See “Configuring netmon” later in this section. -J Add if you have <i>few</i> SNMP agents on your network to accelerate initial discovery.
ovtopmd.lrf	The LRF for ovtopmd, the background process that controls the topology database. Default option: -o which means that it is started through ovstart. If you have purchased the Ingres SQL database, you may want to use the -s option to enable use of the Ingres SQL database. For information about configuring the Ingres database, see the “Configuration” chapter in the <i>HP OpenView Data Management Administrator's</i> manual.
ovwdb.lrf	The LRF for ovwdb, the background process that controls the OVW object database. Default option: -o which means that it is started through ovstart. You may want to use the -n <number of objects> to reduce the amount of memory used by the ovwdb when managing large number of nodes. See “Configuring ovwdb” later in this section.
snmpCol.lrf	The LRF for snmpCollect, the background process that collects MIB data. Default option: None

LRF	Description
snmpd.lrf	The LRF for snmpd, the agent background process that responds to SNMP requests. This LRF is used when snmpd is listening to the SNMP port (port 161). Default option: -n (this option is required when ovspmd starts snmpd)
trapd.lrf	The LRF for trapd, the background process that multiplexes and logs events. This LRF is used when trapd is listening to the SNMP-trap port (port 162). Default option: None You may use the -s option to change the maximum size of the trapd.log and trapd.log.old files.
ovactiond.lrf	The LRF for ovactiond, the background process that executes configured commands when an event is received. Default option: None You may use the -t option to change the time period that ovactiond will wait for commands to complete.

Configuring netmon

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

The first time you start the HP OpenView network management system, **netmon** automatically generates a map of your network. The map's initial management region contains only the networks directly connected to the management station. You may expand this initial management region two ways:

- Interactively expand the management region using the graphical map's editing operations. Normally, the initial map is expanded by interactively managing the management node's neighboring networks. When you perform the **Options: Manage Objects** operation on a network, its neighboring networks and the nodes within the neighboring networks are discovered and added to the map.
- Optionally, provide a "seed file" for the automatic map generation. If you want the initial map to already include specific networks and nodes in your administrative domain, you can create a seed file for the **netmon** process. The seed file contains a list of nodes in your administrative domain and can accelerate generation of your network map by defining a more complete initial management region. The contents of the file define your map's initial management region; neighboring networks that are automatically discovered will appear on the map as "unmanaged" (unmonitored).

A seed file is also useful if your administrative domain is not adjacent to the management system. Normally, **netmon** generates the map outward from the management system. A seed file can force **netmon** to generate the map starting from other nodes instead.

Seed File Format

The format of the seed file is a list of host names or IP addresses of SNMP nodes within your administrative domain. Gateways make the best seeds. You do not have to include all nodes in the seed file. If you add the major gateways, **netmon** will discover the rest of the nodes on the network. Each host name or IP address must be on its own line and all addresses listed must support SNMP. Specifically, **netmon** must be able to determine the subnet mask for addresses of the node's interfaces, otherwise, the seed file entry will be ignored. Comments (preceded by a # sign) are allowed after the host name or IP address on a line. Networks implied by the nodes' IP addresses are added automatically to the map. You choose the name and location of the seed file.

EXAMPLE Seed File:

```
node1.division.company.com
router4.division.company.com      #Gateways make the best seeds
192.0.2.44
192.0.2.45
```

After you have created the seed file, you must modify the `/usr/OV/lrf/netmon.lrf` file so that **netmon** takes the seed file as input. These steps assume that the **netmon** process is running, for example, you followed the default installation which automatically starts all the background daemons. If you are installing the product for the first time and you are following the manual installation procedure described in the *Installation Guide*, skip the steps marked with an asterisk (*).

1. * Stop the **netmon** process by typing

```
ovstop netmon
```

2. Make a backup copy of the `/usr/OV/lrf/netmon.lrf` file.

If this file does not exist, copy the one from
`/usr/OV/newconfig/OVIPMAP/netmon.lrf` to `/usr/OV/lrf/netmon.lrf`.

3. Load the `netmon.lrf` file into your text editor. The `netmon.lrf` file consists of two lines.

```
netmon: /usr/OV/bin/netmon:
OVs_YES_START: ovtopmd, trapd, ovwdb: -P:OVs_WELL_BEHAVED:15:
```

4. Insert the seed file option in the third field of the second line.

For example, adding a seed file, `/usr/OV/conf/nodelist`, to use as input to `netmon` would change the file to look like:

```
netmon: /usr/OV/bin/netmon:  
Ovs_YES_START:ovtopmd,trapd,ovwdb:-P -s /usr/OV/conf/nodelist: Ovs_WELL_BEHAVED:15:
```

If you are starting `snmpd` through `pmd`, you also need to add `snmpd` to the list of dependencies in the second field of the second line.

5. Save the edited `netmon.lrf` file under the name of `/usr/OV/lrf/netmon.lrf`.
6. * Update the configuration file, `ovsuf`. To do so, run `ovaddobj` on the `netmon.lrf` file by typing:

```
ovaddobj /usr/OV/lrf/netmon.lrf
```

`ovaddobj` changes the `ovsuf` configuration file, the configuration file that tells the start-up process what to start.

7. * Start the `netmon` background process by typing

```
ovstart netmon
```

The steps to modify the `netmon.lrf` file to use a seed file are done.

After initial map generation using a seed file, any expansion of the map takes precedence over the (more limited) contents of the seed file. If you edit the seed file to contain nodes not already on the map, subsequent invocations of `netmon` add these nodes and their corresponding networks to the map.

Configuring ovwdb

`ovwdb` is the daemon that controls the OVW object database. When `ovwdb` starts, it loads all the information (or up to the limit set by the `-n` option) in the OVW object database into memory for fast access. Depending on the number of objects in the OVW object database, this can cause performance problems.

To reduce the amount of memory used by the `ovwdb` process and to improve overall performance, you can modify the `ovwdb.lrf` file. To edit the `ovwdb.lrf` to reduce the amount of memory used by `ovwdb`, follow these steps. Note that these steps assume that HP OpenView Windows and the `ovwdb` background processes are running. These background process are running if you followed the default installation that automatically starts all the background daemons. If you are installing the product for the first time and you are following the manual installation procedure described in the *Installation Guide*, skip the steps marked with an asterisk (*).

1. * If HP OpenView Windows (**ovw**) is running, exit HP OpenView Windows by selecting the **File: Exit** menu item.
2. * Stop the background processes by typing

ovstop

3. Make a backup copy of the **/usr/OV/lrf/ovwdb.lrf** file.

If this file does not exist, copy the one from **/usr/OV/newconfig/OVWIN/ovwdb.lrf** to **/usr/OV/lrf/ovwdb.lrf**.

4. Load the **ovwdb.lrf** into your text editor. The default **ovwdb.lrf** file consists of two lines.

```
ovwdb: /usr/OV/bin/ovwdb:
OVS_YES_START: -O -n 5000:OVS_WELL_BEHAVED:15:
```

5. Modify the following option in the third field of the second line.

-n *<number of objects>*

The *<number of objects>* will depend on two things:

- The number of nodes your HP OpenView network management system will manage.
- The amount of memory resources on your system.

As a general rule, assume three objects in the **ovwdb** database for every node. This means that on a system with a 16 megabytes process size, no more than 5,000 objects should be kept in the database without using the “**-n**” option. (5,000 objects is about 1,500 nodes.)

When the “**-n**” option is set, **ovw** continues to add objects to the OVW object database as long as disk space exists. You should not need to set this option lower. If your HP OpenView network management system has swapping problems managing about 500 to 700 nodes, the “**-n**” option will not help. If you have swapping problems managing 500 to 700 nodes, you can either reduce the load on the system from other usage, or get a larger system.

For an HP-UX Series 700 or 800 system, you may double the number of objects kept in the OVW object database; that is, you may keep 10,000 objects in the OVW object database. By default the “**-n**” option is set to 5,000 objects.

If you manage more nodes than the “**-n**” value, the system accesses the objects from the hard disk.

6. Save the edited **ovwdb.lrf** file under the name **/usr/OV/lrf/ovwdb.lrf**.

7. * Update the configuration file, **ovsuf**. To do so, run **ovaddobj** on the **ovwdb.lrf** file by typing

ovaddobj /usr/OV/lrf/ovwdb.lrf

ovaddobj changes the **ovsuf** configuration file, the configuration file that tells the start-up process what to start.

8. * Restart the background processes by typing

ovstart

The steps to modify the **ovwdb.lrf** file to limit the number of objects kept in memory are done.

Configuring the SNMP Management Station

To configure SNMP values for SNMP applications, use the **Options: SNMP Configuration** menu item. The **SNMP Configuration** menu item enables you to change the default values for agent community names, timeout and retry intervals, proxies, remote ports, and status polling intervals. You can also change these values using a command line interface. For more information on this see the **snmpconf(1)** man page.

You can also configure a specific node or a group of nodes to have different values than the default configuration. You configure a group of nodes by specifying an IP address wildcard (for example, **15.122.*.***). The IP address wildcard is useful, for example, when you want to configure different timeout values and number of retries for Wide Area Networks (WANs).

For information about how to use the **SNMP Configuration** operation, see the online help. For technical information, see the **ovsnmp.conf(4)** man page.

Proxy Configuration

You can use a proxy to allow SNMP access to nodes which do not support SNMP. When you configure a proxy, the proxy agent receives the SNMP request and forwards it to the requested node using a non-SNMP protocol. How the proxy gets information from the target node depends on the target.

EXAMPLE:

Let's say you want to get information about a LAN Manager/X client, which is a PC node. You cannot get information directly from the PC node because the PC does not support SNMP. However, the LAN Manager/X server supports SNMP and the server can communicate with the PC node. In this example, you can configure the LAN Manager/X server to act as a proxy for the target PC node. All requests to the target PC node are really sent to the server.

Community Configuration

This is the SNMP community name that the management application uses as authorization when issuing SNMP GetRequests and GetNextRequests to the managed node.

This community name is also used for SNMP SetRequests if the management application cannot resolve an explicit Set Community name for the managed node.

The community name configuration on the management station must match the value of the community name configured on the agent. For instructions on how to configure the agent community name, see the documentation provided by the vendor of the agent. For example, for HP OpenView agents, see the *HP OpenView SNMP Agent Administrator's Reference* manual.

Set Community Configuration

This is the SNMP community name that the management application uses as authorization when issuing SNMP SetRequests to the managed node.

Often times, SNMP agents are configured with different community names for SetRequests than for GetRequests. This field enables you to configure the SNMP management applications according to this distinction.

Timeout and Retry Configuration

To do effective polling, you may want to change the timeout and retry values. For example, latency times are greater over WANs than LANs. To prevent the management station from a premature time out when making requests across a WAN, you may want to use the **Options: SNMP Configuration** menu item to increase the Timeout and Retry values.

Remote Port Configuration

You can specify the UDP port number on the managed node or proxy node (as appropriate) where the SNMP agent expects to receive SNMP requests. The standard SNMP port is 161.

You generally specify a remote port only for specialized proxy agents that do not listen to the standard SNMP port.

netmon Configuration

You also use the **Options: SNMP Configuration** menu item to change **netmon** status polling intervals. The community name, timeout, and retry interval are also used by **netmon** for SNMP related operations (that is, configuration and discovery polls). These fields are also used for ICMP operations (that is, status polling). To configure other **netmon** configuration parameters such as frequency of discovery, poll operations, address table poll operations, timeout intervals, retries, proxies, and configuration check operations, use the **Options: Topology/Status Polling: IP** menu item.

Configuring the Mapping from sysObjectID to Types

The nodes in your network are represented in HP OpenView Windows (ovw) as symbols. The HP OpenView network management system displays these symbols based on mappings from the **sysObjectID** to OVW symbol types and vendor, SNMP agent, and IP topology attributes. This mapping is done through two configuration files.

- **oid_to_sym**. The **oid_to_sym** configuration file is used by **ipmap**. When **ipmap** puts a symbol on the map, it tries to match the **sysObjectID** to a symbol type to be used in the map.
- **oid_to_type**. The **oid_to_type** configuration file is used by **netmon** to map the **sysObjectID** of a node into the correct IP topology behavior and to the correct vendor and SNMP agent values.

This section describes how you can edit these files so that the nodes in your network will be represented with an appropriate symbol in HP OpenView Windows. For information about **sysObjectID**, see chapter 1.

Editing the oid_to_sym Configuration File

The **/usr/OV/conf/\$LANG/oid_to_sym** configuration file specifies a mapping from the **sysObjectID** of an agent to a default symbol class and subclass (that is, to the OVW symbol type). The HP OpenView network management system chooses an appropriate symbol type to represent nodes in the IP topology maps in HP OpenView Windows based on the value of **sysObjectID**. The **oid_to_sym** file shipped with the HP OpenView product contains entries for various agents. You can edit these entries or add new ones as desired.

Note If **\$LANG** is not set, it defaults to **c**.

To edit the **oid_to_sym** file, follow these steps:

1. Exit HP OpenView Windows (ovw) by selecting the **File: Exit** menu item.
2. Make a backup copy of the **/usr/OV/conf/\$LANG/oid_to_sym** file.

Note that the original version of the **oid_to_sym** file is installed in **/usr/OV/newconfig/OVIPMAP/oid_to_sym** should you need it.

3. Load the **oid_to_sym** file into your text editor.

4. Edit the `oid_to_sym` file. See the field descriptions below.
5. Save the edited `oid_to_sym` file.
6. Type the following command for each `sysObjectID` that changed symbol types.

```
ovtopofix -u -o sysObjectID
```

7. Restart HP OpenView Windows by typing the `ovw` command.

Each entry in the `oid_to_sym` file consists of three fields separated by a colon (:). Here are some examples from the default `oid_to_sym` file.

```
# HP unix-based Agents
1.3.6.1.4.1.11.2.3.2.2:Computer:Workstation # series 300
1.3.6.1.4.1.11.2.3.2.3:Computer:Mini # series 800
1.3.6.1.4.1.11.2.3.2.5:Computer:Workstation # series 700
1.3.6.1.4.1.11.2.3.8:Computer:PC # HP 386 agent
1.3.6.1.4.1.11.2.3.10.1:Computer:Workstation # HP SunOS Sparc agent
```

- The first field is the MIB object reported by an agent as the value of the Internet-standard MIB-II `system.sysObjectID`, for example, `1.3.6.1.4.1.11.2.3.2.2`.
- The second field is the default symbol class. Valid symbol class entries include Connector, Computer, Device, or other OVW symbol class entries. The default value is Computer.
- The third field is the default symbol subclass. Valid symbol subclass entries for the Computer class include Workstation, Mini, PC, and Mainframe.

The values for symbol class and subclass must match one of the registered symbol types with `ovw`. To find out about the currently defined symbols, see the `/usr/OV/symbols/$LANG` registration files or **Help: Display Legend**. To find out how to create new symbols, see the `OVwRegIntro(5)` man page.

The class and subclass fields together make up the OVW symbol type. The number symbol (#) indicates the beginning of a comment. Blank lines are ignored.

EXAMPLE:

Let's say you want to change the symbol for an HP 9000 Series 300 agent from the default symbol subclass Workstation to a minicomputer. To do so, edit the following line

```
1.3.6.1.4.1.11.2.3.2.2:Computer:Workstation # 300
```

to look like

```
1.3.6.1.4.1.11.2.3.2.2:Computer:Mini # 300
```

When you add new symbol types you can add or change entries in the `oid_to_sym` file to take advantage of the new symbols. (See the "Creating and Using Registration Files" appendix in this manual.) However, the symbol class and subclass must always match the list of symbols supported by `ovw`. If they do not match, then `ipmap` may be unable to add the symbol to the IP topology map.

Editing the oid_to_type Configuration File

The `/usr/OV/conf/oid_to_type` configuration file specifies a mapping from the `sysObjectID` of an agent to the correct IP topology behavior and to the correct vendor and SNMP agent values. The `oid_to_type` file shipped with the HP OpenView product contains entries for various enterprise-specific MIBs which are generally available. You can edit these entries or add new ones as desired.

To edit the `oid_to_type` file, follow these steps:

1. Stop `netmon` by typing

```
ovstop netmon
```

2. Make a backup copy of the `/usr/OV/conf/oid_to_type` file.

Note that the original version of the `oid_to_type` file is installed in `/usr/OV/newconfig/OVIPMAP/oid_to_type` should you need it.

3. Load the `oid_to_type` file into your text editor.
4. Edit the `oid_to_type` file. See the field descriptions below.
5. Save the edited `oid_to_type` file.
6. Restart `netmon` by typing

```
ovstart netmon
```

Each entry in the `oid_to_type` file consists of up to four fields separated by a colon (:). Here is an example from the default `oid_to_type` file.

```
1.3.6.1.4.1.23.1.1.1:Novell:Novell Lantern
1.3.6.1.4.1.9.1.5:cisco Systems:Cisco Gateway Server:GS
1.3.6.1.4.1.26.2.1:Hughes:Hughes LAN Systems Bridge:B
```

- The first field is the MIB object reported by an agent as the value of the Internet-standard MIB-II `system.sysObjectID`, for example, `1.3.6.1.4.1.23.1.1.1`.
- The second field is the vendor which manufactures the given device or node. The vendor name must match one of the enumerated values in the field registration file `/usr/OV/fields/$LANG/ovw_fields` for the “vendor” field. Examples include Hewlett-Packard, IBM, Sun, and DEC. If the vendor name does not match a valid value, it gets set to “unset.” To add new values for the “vendor” field, see the next section “Adding Values for vendor and SNMP Agent Fields.”

- The third field is the agent type currently running on the given device or node. The agent name must match one of the enumerated values in the field registration file `/usr/ov/fields/$LANG/snmp_fields` for the “SNMP Agent” field. The typical agent name includes the manufacturer of the agent software and the type of device on which the agent software runs. Examples include “HP 9000/HP-UX,” “HP 3000/XL,” and “HP SunOS Sparc”; the latter is the HP supported agent that runs on a SunOS system. To add new “SNMP Agent” field values, see the next section “Adding Values for vendor and SNMP Agent Fields.”
- The fourth field is a set of flags controlling what topology SNMP attributes should be applied to an object with this `sysObjectID`. The topology attributes field controls how the device will be treated by the IP-specific components of the system. These are only default attributes; if it can be determined that a device is or is not performing any of these roles, the attributes for that particular device will be set correctly. Specifically, any combination of flags may be applied to a `sysObjectID`. See table 3-2.

You can also specify one of two keywords instead of specifying the first field:

DEFAULT_IP matches any `sysObjectID` returned by an agent
DEFAULT_SNMP used for systems without SNMP agents

Table 3-2. SNMP Topology Attributes

Flag	Meaning
G	Treat the device topologically as a gateway (router). A symbol for this object will appear in the Segment, Network, and Internet submaps, and the symbol can be used to connect networks.
B	Treat the object as a bridge or simple repeater. A symbol for objects with this <code>sysObjectID</code> will appear in the Segment and Network submaps, and the symbol can be used to connect segments.
H	Treat the object as a multiport repeater or hub. A symbol for objects with this <code>sysObjectID</code> will appear in the Segment and Network submaps, and the symbol can be used to connect segments. Also, this symbol can appear at the center (hub) of Star segments.
I	Ignore the node's ability to support SNMP.
T	Report the node's address as if it were a terminal server.
S	Treat the object as a device that supports secondary IP addresses for interfaces. If this flag is not present, interfaces can continually get deleted and rediscovered.
U	Automatically unmanage nodes of this type upon discovery. You will still be able to manually manage the node. It is useful to use this option in conjunction with the DEFAULT_IP capability.

EXAMPLES:

Let's say you have an entry in the `oid_to_type` file for an HP Router that looks like

```
1.3.6.1.4.1.11.2.3.7.2:Hewlett-Packard:HP Router:G
```

This entry indicates that the vendor is "Hewlett-Packard," the agent type is "HP Router," and the device acts as a gateway in the IP topology.

```
DEAFULT_IP:::U
```

Specifies that any system that does not respond to SNMP will be created as an unmanaged node.

You can access the "vendor" and "SNMPAgent" fields associated with an object by selecting the **Monitor: Description -> Selected Objects** menu item and view the **General Attributes** option in the **Object Attributes** section of the dialog box. The "SNMPAgent" field is particularly important since it is used as a capability for determining which OVW application menu items will be enabled for that object.

Note

Entries for certain Hewlett-Packard devices are preconfigured into the system, and you cannot change them. This is necessary because **netmon** takes advantage of certain characteristics of these devices. These entries are specified through a separate configuration file, `/usr/ov/conf/HPoid2type`. Changes made to this file could be overwritten during product updates and installation. HP does not support editing of this separate configuration file.

Adding Values for vendor and SNMPAgent Fields

To define additional values for the “vendor” and “SNMPAgent” fields, follow these steps.

1. Create a file with definition extensions.

Let's say you work for the Flintstones Company and you want to add three entries to the “SNMPAgent” field. To do so, create a new file with the following information:

```
Field "SNMPAgent" {
    Type Enumeration;
    Flags    capability, general, locate;
    Enumeration "Unset",
        "Fred Router",
        "Wilma Hub",
        "Barney Bridge"
    ;
}

Field "vendor" {
    Type Enumeration;
    Flags    capability, general, locate;
    Enumeration "Unset",
        "Flintstones Company"
    ;
}
```

Note that everything except for the values is the same as what is defined in the standard HP OpenView product. The first four lines of the “SNMPAgent” field are identical to the definition lines in the `snmp_fields` file, and the first four lines of the “vendor” field are identical to the definition lines in the `ovw_fields` file.

For more details about the syntax, see the “Creating and Using Registration Files” appendix.

2. Save your new file in the `/usr/ov/fields/c` directory.
3. As root, add the new definitions to the OVW object database by typing

```
ovw -fields
```

The steps to add values for “vendor” and “SNMPAgent” fields are done. The new definitions appear in the **General Attributes** selection of the **Describe/Modify Object** operation and can be referenced in `oid_to_type` mappings described in “Editing the `oid_to_type` Configuration File.”

Configuring Terminal Connectivity to Remote HP 3000s

The **Misc: Terminal Connect -> vt3k (hp3000)** operation allows you to log into remote HP 3000 systems. The **vt3k** operation requires the HP Network Services (NS) node name of the remote HP 3000. However, nodes selected from the map are identified by their ARPA Services host names. By default, a selected node's ARPA host name is truncated after the first . (dot), and the first part of the ARPA host name becomes the NS node name input to the **vt3k** operation. This procedure assumes that the truncated name identifies a node in the same NS domain as the management system, since a fully-qualified NS node name is unavailable.

The **vt3k** operation can fail if your nodes' ARPA host names differ from their NS node names or if you are trying to connect to HP 3000 nodes outside of your NS domain. If this is happening, you need to configure the `/usr/OV/conf/xmmvt3k.conf` file. This file associates HP 3000 ARPA host names with corresponding NS node names. A script called `/usr/OV/bin/xmmvt3k` actually performs the name translation using the following rules when you select the

Misc: Terminal Connect -> vt3k (hp3000) operation. You can select either Block Mode or Type Ahead mode.

The **vt3k** operation resolves the ARPA host name to NS node name as follows:

1. Search for the first line in the `/usr/OV/conf/xmmvt3k.conf` file that begins with a matching ARPA host name. Use the NS node name in the next column as input to the **vt3k (hp3000)** operation.
2. If no matching ARPA host name is found in the `/usr/OV/conf/xmmvt3k.conf` file, repeat the search using only the first part of the ARPA host name (the part preceding the first dot). If a matching name is found, use the NS node name in the next column as input to the **vt3k (hp3000)** operation.
3. If no matching name is found in the `/usr/OV/conf/xmmvt3k.conf` file (the default case), use the truncated ARPA host name as input to the **vt3k (hp3000)** operation. This case assumes that the name identifies a node in the same NS domain as the management system, since a fully-qualified NS node name is unavailable.

You may configure the `/usr/OV/conf/xmmvt3k.conf` file at any time; you do not have to exit and restart any part of the HP OpenView product. See the example `xmmvt3k.conf` file on the next page.

Edit the `/usr/ov/conf/xxmvt3k.conf` file as follows:

- Begin each line with the ARPA host name of one of your HP 3000 systems.
- After each ARPA host name, put its corresponding NS node name on the same line, separating the two names by any number of tabs or spaces.
- Precede any comments with a pound (#) sign.

EXAMPLE:

```
myhost    mynode    # My local HP 3000
mailhub.department.university.edu mailhub.department.university # HPDESK Hub
admin.division.company.com    finance.domain.organization # Corporate HP 3000
```

Integrating Existing Applications

You may add your own application to HP OpenView Windows (OVW) and specify that it be invoked from an HP OpenView Windows menu item. To do this, you should have an understanding of integration concepts.

Application Integration with HP OpenView Windows

An application can be integrated into HP OpenView Windows in one of two ways.

- Integration through the HP OpenView Windows menu bar only.

Integration is provided through an Application Registration File, which defines how users access applications and how application processes are managed. Applications added through the menu bar are also available as a choice for an executable symbol. For conceptual information, as well as information on how to create executable symbols, see the *HP OpenView Windows User's Guide*.

- Programmatic integration with HP OpenView Windows.

In addition to the Application Registration File, the application also has programmatic access to HP OpenView Windows through the HP OpenView Windows Application Programming Interface (OVW API).

This document covers only the first case above, as the second case applies to application developers who have purchased the HP OpenView SNMP Developer's Kit.

Creating an Application Registration File

Every application that is integrated into the HP OpenView Windows (OVW) menu bar has an Application Registration File. This file contains fields that define the menu bar items from which the application will be accessed, and fields that specify the command line that invokes the application.

This section gives an example of how to create an Application Registration File. For detailed information, see this manual's "Creating and Using Registration Files" appendix or the **OVWRegIntro(5)** man page.

EXAMPLE:

Assume you want to integrate UNIX commands into the OVW menu bar. This example integrates two commands into one OVW application. One command looks at the mail queue on a network node and the other continually monitors the mail log on a network node.

The name of your application is "Mail Manager" and the registration file name is **mail-reg**.

1. Create a file in the **/usr/OV/registration/C** directory called **/usr/OV/registration/C/mail-reg**.
2. Load the **mail-reg** file into your text editor.
3. Specify the name of the application, in this case Mail Manager, and define an Application block by typing:

```
Application "Mail Manager"
{
}

```

4. Define the menu structure by adding MenuBar and Menu blocks within the Application block. You need to have menu items in the OVW menu bar to integrate the UNIX commands into the OVW menu bar.
 - a. In the MenuBar block, specify the OVW main menu bar under which your UNIX commands will be integrated and the name of the menu item. In this example, the menu item is **Mail** and it will be under the **Monitor** menu bar item.
 - b. In the Menu block, specify the menu items that will execute your UNIX commands. In this example the menu items are **Mail Queue** and **Mail Log**.

After this step your **mail-reg** file will look like this.

```
Application "Mail Manager"
{
    MenuBar "Monitor"
    {
        "Mail" f.menu "MailMenu";
    }

    Menu "MailMenu"
    {
        "Mail Queue" f.action "Mailq";
        "Mail Log" f.action "MailLog";
    }
}

```

5. Define the actions that will happen when the menu items are selected. In this example, the menu items will only be invoked on one object that has the "isNode" capability enabled.

Your **mail-reg** Application Registration File is created. The finished file looks like this.

Application "Mail Manager"

```
{

  MenuBar "Monitor"
  {
    "Mail" f.menu "MailMenu";
  }

  Menu "MailMenu"
  {
    "Mail Queue" f.action "Mailq";
    "Mail Log" f.action "MailLog";
  }

  Action "Mailq"
  {
    SelectionRule isNode;
    MinSelected 1;
    MaxSelected 1;
    /* Note, you can use remsh instead of rexec */
    Command 'xterm -title "${OVwMenuItem} (${OVwSelection1})" \
      -e sh -c "/usr/bin/rexec ${OVwSelection1} \
        -l root /usr/bin/mailq ; \
        echo ; echo \'Press return...\' ; read line"';
  }

  Action "MailLog"
  {
    SelectionRule isNode;
    MinSelected 1;
    MaxSelected 1;
    /* Note, you can use remsh instead of rexec */
    Command 'xterm -title "${OVwMenuItem} (${OVwSelection1})" \
      -e sh -c "/usr/bin/rexec ${OVwSelection1} \
        -l root tail -f /usr/spool/mqueue/syslog"';
  }

}
```

6. Save the registration file, start **ovw**, select a node, and try the new menu items. If **ovw** is already running, then exit out of **ovw** using the **File: Exit** menu item and then restart **ovw**.

Modifying an Existing Application

You may have an application installed into HP OpenView Windows (OVW) that you would rather have under a different menu bar.

To modify that application's Application Registration File and change the menu bar location of the application, follow the guidelines given in the previous section "Creating an Application Registration File."

For the changes to take effect, restart **ovw**.

Automatic Start-Up of Your Application

HP OpenView Windows invokes an application at run-time when you select the appropriate menu item. Each selection of that menu item causes another instance of the application to be loaded and executed. You can change this behavior by adding 'Flags' to the command statement in the Application Registration File.

There are three command "flags":

- **Initial.**

Tells OVW to start the application when OVW is started. The application will be invoked by the "Command" line in the application block.

- **Shared.**

Tells OVW that only a single instance of the command should be running at any time. Further, the command will be shared, servicing multiple action requests from the user interface. (This requires programmatic use of the **ovw** API.)

- **Restart.**

Tells OVW to restart the application if it should ever exit. This flag is intended for applications that are required for normal OVW operation, such as those managing the semantics of OVW maps.

You add the flags after the “command” keyword as follows:

```
Application "My App"  
{  
..  
Command -Initial -Shared -Restart "/usr/OV/bin/myapp";  
..  
}
```

You may specify one or more flags on this line. For more information, see the “Creating and Using Registration Files” appendix in this manual or the **OVwRegIntro(5)** man page.

Changing File or Map Permissions

In HP OpenView Windows (OVW), user access to maps is governed by the UNIX permissions of the individual map. You may want to change these permissions for security reasons.

- Use **ovwchown** to change the ownership of individual maps.
- Use **ovwchmod** to change the mode of individual maps.
- Use **ovwchgrp** to change the group.

You can use the **ovwls** command to list the available maps and their current permissions.

For details on **ovwchown**, **ovwchmod**, and **ovwchgrp**, see this manual's "Reference" appendix. Note that all three commands (along with the **ovwls** command) are documented on the **ovwperms (1)** man page.

You must be superuser or the owner of the target map to perform these tasks.

Note Do not change map permissions when OVW is running.

To change permissions of other HP OpenView configuration files and database files, use the standard **chown**, **chmod**, and **chgrp** commands.

Defining New Symbol Classes

HP OpenView Windows is shipped with a set of predefined symbols. If these predefined symbols do not meet your needs, you can define your own symbols. For information about how to do this, see the “Creating and Using Registration Files” appendix in this manual.

On-Going System Maintenance

To optimize the management station's performance, you may need to perform the following tasks.

File Clean-Up

snmpCollect/

Maintain the data collection directory, **/usr/ov/databases/snmpCollect**. The **snmpCollect** process saves all collected data to this directory. The directory continues to grow as long as you are collecting data. To prevent the **snmpCollect** directory from filling up all available disk space, do the following:

- Reduce the polling intervals. Using the **MTB Data Collection** operation, set the polling interval to not collect as often. In addition, if you only want to check thresholds, don't store the data. See the *HP OpenView Network Node Manager User's Guide*.
- Set up **cron** (**1M**) to periodically remove files in the **/usr/ov/databases/snmpCollect** directory and the log file **/usr/ov/bin/log/snmpCol.trace**.
- To get the last 100 entries of a file in the **snmpCollect** directory, use the following commands:

```
snmpColDump -tTI /usr/ov/databases/snmpCollect/FILE | \
    awk -F\t '{printf("%d\t%d\t%s\t%lg\n", $4, $5, $6, $3)}' | \
    tail -100 > /tmp/save

snmpColDump -r /tmp/save /usr/ov/databases/snmpCollect/FILE
```

ovsuf

Delete unused entries from the `/usr/OV/conf/ovsuf` configuration file. The `ovsuf` file contains the configuration information that tells the start-up process what background processes to start.

You should not normally edit this file. Entries in the `ovsuf` file are added or deleted through the `ovaddobj` or `ovdelobj` commands. However, the `ovdelobj` command does not physically delete an entry in the `ovsuf` file; the entry is still in the file, but the entry has a special designation of “1:” at the beginning of the line. This tells the start-up process not to start this particular entry.

If you have repeatedly been running the `ovaddobj` or `ovdelobj` commands, the file may grow large and you may want to clean up the file. Cleaning up this file consists of deleting all lines that begin with “1:”, which designates a deleted entry.

Here is an example of the `ovsuf` file.

```
1:netmon:/usr/OV/bin/netmon:OVs_YES_START:ovtopmd,trapd,ovwdb:-P:OVs_WELL_BEHAVED:15:
0:ovtopmd:/usr/OV/bin/ovtopmd:OVs_YES_START:trapd,ovwdb:-O:OVs_WELL_BEHAVED:15:
0:ovwdb:/usr/OV/bin/ovwdb:OVs_YES_START::-O:OVs_WELL_BEHAVED:15:
0:trapd:/usr/OV/bin/trapd:OVs_YES_START::-O:OVs_WELL_BEHAVED:15:
0:snmpCollect:/usr/OV/bin/snmpCollect:OVs_YES_START:trapd,ovwdb,ovtopmd:OVs_WELL_BEHAVED:120:
```

Setting Up Automatic Map Snapshots

The HP OpenView Windows graphical user interface has a **File: Map Snapshot -> Create** menu item that enables you to take snapshots of your maps.

Alternatively, you can use the `ovmapsnap` command from a `crontab` entry to automatically take snapshots at regular intervals. For information about `crontab`, see the `crontab(1)` man page.

EXAMPLE:

The following `crontab` entry causes a snapshot, identified by the comment “**Automap**”, to be taken at 1:00 am every Saturday.

```
0 1 * * 6 ovmapsnap -c -C "Automap" -m mapname | mailx -s"Auto Snapshot" ovwuser
```

This example would send `ovwuser` a mail reminder that a snapshot had been taken. `mapname` is the name of the map. Since this process creates snapshots automatically, delete old snapshots periodically. See the next section “Removing Old Snapshots.”

Removing Old Snapshots

The HP OpenView Windows graphical user interface has a **File: Map Snapshot -> Delete** menu item that enables you to remove old snapshots to free up memory and improve system performance.

Alternatively, you can remove snapshots through a terminal interface using commands. To do so, follow these steps:

1. Get a listing of all current maps by typing:

```
ovmapdump -l
```

This will produce a list similar to the following:

MAP	PERMS	CREATION TIME	COMMENTS
default	R/W	Tue Apr 27 11:52:32 1993	
Example Map 1	R/O	Thr Apr 22 12:05:48 1993	example map
Example Map 2	None	Wed Apr 28 14:37:49 1993	

2. List all available snapshots for a given map by using the following command:

```
ovmapsnap -l -m mapname
```

where *mapname* is the name of the map. This will produce a list similar to the following:

NAME	CREATION TIME	COMMENTS
Testing	Fri Apr 23 12:05:48 1993	testing ovmapsnap

3. Delete the given snapshot by using the following command:

```
ovmapsnap -d -n Testing -m mapname
```

If you want to automatically delete the oldest entry, set up a **crontab** entry using the following command:

```
ovmapsnap -d -f -m mapname
```

Note that this command automatically deletes the oldest entry and that may not be the entry you wanted to delete. See the **ovmapsnap(1)** man page for more details.

Backing Up HP OpenView

You should make a weekly backup of the HP OpenView databases and log files. You should always back up before any of the following situations:

- When preparing to greatly expand the managed area of your network.
- When preparing to do extensive editing and customization of your network map.
- After initial discovery or expansion of the discovery process when you feel you have a map that accurately reflects your network.

To back up HP Openview databases and log files, follow these steps:

1. Exit any executing **ovw** sessions using the **File: Exit** menu item.
2. Stop the executing daemons by typing

```
/usr/OV/bin/ovstop
```

3. Use your preferred method to back up the HP OpenView databases and log files. To do this, you must know the file name of the device file that you are going to back up to.

EXAMPLE:

```
cd /usr/OV  
tar cvf device_filename databases log
```

where **device_filename** is the file name for the device you are backing up to.

For example,

```
tar cvf /dev/rmt/0m
```

Note

Do not back up your system without executing **ovstop** first. Backing up an active database can result in a corrupt backup.

4. Restart the background processes by typing:

```
ovstart
```

5. Restart **ovw**.

Restoring HP OpenView From a Backup

To restore HP OpenView databases and log files from a backup, follow the steps below. Note, that you will lose any discovered data and map customization changes since the backup was created.

1. Exit any executing **ovw** sessions using the **File: Exit** menu item.
2. Stop the executing daemons by typing:

```
/usr/OV/bin/ovstop
```

3. Use your preferred method to restore the HP OpenView databases and log files. To do this, you must know the file name of the device file that you are restoring from.

EXAMPLE:

```
cd /usr/OV  
tar xvf device_filename
```

where **device_filename** is the file name for the device you are restoring from.

For example,

```
tar xvf /dev/rmt/0m
```

Note

Do not restore your system without executing **ovstop** first. Restoring a backup onto an active database can result in a corrupt database.

4. Restart the background processes by typing

```
ovstart
```

5. Restart **ovw**.

Troubleshooting

This chapter focuses on troubleshooting the HP OpenView Network Node Manager product itself. It does not cover how to do network troubleshooting.

This chapter discusses the following topics:

- recommended practices for problem prevention, isolation, and recovery
- characterizing the problem
- general product troubleshooting considerations
- troubleshooting activation key problems
- troubleshooting IP discovery and layout
- troubleshooting background processes
- troubleshooting various operations
- performance and traffic

Recommended Practices

Following these recommended practices helps you to prevent problems, isolate problems, and recover from problems.

- Ensure that the management system meets the hardware, software and configuration prerequisites, and recommendations. Refer to the *Installation Guide* for hardware and software prerequisites. Chapter 3 of this manual contains configuration prerequisites information.
- Maintain the system by regularly editing files and deleting files and snapshots. See the recommendations in the “On-Going System Maintenance” section in chapter 3.
- Ensure that `/usr/OV/bin` is in your path.
- Use the **File: Save Map As** menu item to save copies of the map. See the *HP OpenView Windows User's Guide*.
- Do not modify HP OpenView product files, such as X resource files, without retaining the original files. The original files provide a way of restoring a known good operational configuration. If you correct a problem by restoring the original files, you can isolate the problem to the changes you made to these files.

When you back up our original files, do not copy the files to a new file name in the same directory. All files in the following directories are considered “live,” that is, every file in these directories will be used and you will end up with duplicate definitions.

```
/usr/OV/registration/*  
/usr/OV/fields/*  
/usr/OV/symbols/*  
/usr/OV/bitmaps/*  
/usr/OV/conf/*  
/usr/OV/help/*
```

Therefore, to back up your original files, do one of the following:

- Create a directory called `/usr/OV/backup` and replicate the directories listed above in this directory.
 - Back up the original files to tape or a tar file.
- HP recommends that you not modify the files in `/usr/OV/app-defaults`. Instead, you can override the default X resources by modifying them in your `$HOME/.Xdefaults` file. If you are running HP VUE, modify your view resources. See `/usr/OV/app-defaults/XNm` for more details.

- Make sure that you are not using up too much of your management station's CPU and network resources by collecting too much MIB data or setting too frequent polling intervals for the **netmon** process.
- Use logging and tracing of background processes to help isolate problems, but be sure to clean up log and trace files regularly.
- Use **ovstatus**, and the **-v** option to **ovstart**.

Logging and Tracing

The HP OpenView network management system uses two logging and tracing facilities. See table 4-1.

Table 4-1. Logging and Tracing Facilities

For this process	Use this facility
trapd netmon ovactiond snmpCollect	command options for the background processes
ipmap ovwdb ovtopmd ovspmd	nettl for logging and tracing netfmt to format and view the logged data

Note

Tracing is normally used only by your support personnel. Tracing, especially of **netmon** and **snmpCollect**, can cause the trace file to grow extremely large (multiple megabytes in size) fairly quickly. If you use tracing, remember to monitor the size of the trace file frequently, and turn tracing off as soon as you are finished.

Using Command Options

The following section describes logging and tracing options for HP OpenView Network Node Manager background processes **trapd**, **netmon**, **snmpCollect**, and **ovactiond**.

Using trapd, netmon, ovactiond, and snmpCollect

The **/usr/OV/bin/trapd** logging facility has the following characteristics:

- The default logging is set to ON.
- The log file is **/usr/OV/log/trapd.log**. When this log file is filled to a user configurable size, its contents are automatically moved to the file **/usr/OV/log/trapd.log.old**.

The **/usr/OV/bin/trapd** tracing facility has the following characteristics:

- The default tracing is set to OFF.
- The default trace and hexdump file is **/usr/OV/log/trapd.trace**.
- To turn tracing on and off, specify **trapd -T** (toggle on/off).
- To turn hexdump of traps on, specify **trapd -x**.
- To turn hexdump off, kill and restart the process.

The **/usr/OV/bin/netmon** tracing facility has the following characteristics:

- The default tracing is set to OFF.
- The default trace and hexdump file is **/usr/OV/log/netmon.trace**.
- To turn tracing on, specify **netmon -M tracemask** or **netmon -m tracemask** at invocation.
- To turn tracing off, specify **netmon -M 0** (zero).

The **/usr/OV/bin/ovactiond** tracing facility has the following characteristics:

- The default tracing is set to ON.
- The default tracing file is **/usr/OV/log/ovactiond.log**.
- To turn verbose tracing on and off when tracing is on, specify **ovactiond -v** (toggle on/off).
- The **-t** option must have a value greater than 0 (zero) to see the return value of the event actions.

The **/usr/OV/bin/snmpCollect** logging facility has the following characteristics:

- The default logging is set to ON.
- The data that is collected is stored in the directory **/usr/OV/log/snmpCollect**.

The `/usr/OV/bin/snmpCollect` tracing facility has the following characteristics:

- The default tracing is limited.
- The default trace file is `/usr/OV/log/snmpCol.trace`.
- To turn the tracing on and off, specify `snmpCollect -T` (toggle on/off).
- To turn verbose tracing on and off when tracing is on, specify `snmpCollect -v` (toggle on/off).

The following information explains the variable `tracemask` discussed in the `trapd` and `netmon` tracing facilities. Trace masks specify the type of output listed in `netmon`'s trace file. To select multiple output types, add the individual `tracemask` values together and enter the sum. `netmon` has the following tracing options:

0	Turn off tracing.
1	Trace ICMP echo requests.
2	Trace ICMP echo replies and timeouts
4	Trace SNMP requests.
8	Trace SNMP replies and timeouts.
16	Trace traps generated.
32	Trace traps received.

EXAMPLES: (These assume `netmon` is already running.)

To trace ICMP echo replies and timeouts, enter the following command:

```
netmon -M 2
```

To trace ICMP echo replies and timeouts, as well as, SNMP replies and timeouts, add their `tracemask` values (2 + 8) to determine the `tracemask` value to enter.

```
netmon -M 10
```

Using *nettl* and *netfmt*

nettl and *netfmt* enable system administrators and developers to gather and view data about the operation of HP OpenView software.

- *nettl* is the administrative interface to two facilities:
 - **logging**, which records externally-observable events such as connection establishment or termination of child processes, and errors, which daemon processes ordinarily have no other way of reporting.
 - **tracing**, which records a detailed record of internal events taking place within the software. Tracing of the HP OpenView daemons is generally only used on the advice of HP Support to diagnose difficult problems, with the aid of the source code. API tracing is used by developers to determine whether they are using the HP OpenView APIs correctly.
- *netfmt* formats the log and trace output into human-readable form.

It is recommended that you run the *nettl* logging facility at all times. This way, you can view the constantly-updated log file to help you determine what happened when something goes wrong. Also, paradoxically, performance is better when logging is turned *on*.

End-users should run tracing only when something is seriously wrong, and only on the advice of your support representative, in order to reconstruct in detail the sequence of events that caused the problem. The trace output is intended to be interpreted by the developers of the HP OpenView product, with the source code at hand. It is not intended to be meaningful to the ordinary user.

Developers may find the tracing in the **OVWAPI** subsystem useful in order to diagnose problems with the **ovw** API.

Processes use *nettl* to log or trace in particular subsystems. To examine the log or trace output of a process, you need to know the name of the subsystem the process uses.

Subsystem	Processes Using this Subsystem	Purpose of Process
OVS	ovspmd	Starts and stops HP OpenView daemons
OVW	ovwdb ipmap	Provides access to OVW object database Automatically draws IP topology maps
OVEXTERNAL	ovtopmd Any third-party-developed software using the ovuTL(3) facility	Maintains the topology database
OVWAPI	Any software using the ovw (HP OpenView Windows) API	

Note The *nettl* and *netfmt* commands are in */etc*.

For example, to run the `nettl -status` command, you would type:
`/etc/nettl -status`

Capturing Logging and Tracing Output with nettl

1. To determine whether the logging/tracing facilities are currently running, type the following command:

```
nettl -status
```

2. If necessary, use the **nettl** command to start the logging/tracing facilities. As **root**, type the following command:

```
nettl -start
```

This command initializes the logging/tracing facilities and starts the default logging. By default, the **OVS**, **OVW**, **OVWAPI**, and **OVEXTERNAL** subsystems log error and disaster messages; the **OVS** subsystem also logs informative messages.

3. Once **nettl** is running, you can use command options to customize the kind of data you want to capture for any of the **nettl** subsystems shown in the previous table. The next section contains examples of various **nettl** commands.

Examples of nettl Commands

Use the following examples to get started. After you have captured the data you want, go to the section, "Formatting Network Data with netfmt".

■ Logging Facility - Turning it on.

There are four classes of logging messages: informative (i), warning (w), error (e), and disaster (d). Default logging always includes disaster-class messages; you can turn on any of the other three classes by typing a **nettl** command for a specific subsystem.

The command syntax is:

```
nettl -log class -entity subsystem OVEXTERNAL
```

-log class	Specifies the classes of logging to be turned on for the specified subsystems.
-entity subsystem	Specifies the subsystems for which logging is to be modified. For logging done by HP OpenView products, OVEXTERNAL must always be in the list.

Example 1:

Turn on logging of informative messages for the **ovw** subsystem. As **root**, type the following command:

```
nettl -log i -entity Ovw OVEXTERNAL
```

Note that this command will turn off any log classes not listed in the command (in this example, the warning and error messages would be turned off). The one exception is disaster logging, which cannot be turned off.

Example 2:

Turn on logging of informative, warning, and error messages for the **ovs** subsystem. As **root**, type the following command:

```
nettl -log i w e -entity OVS OVEXTERNAL
```

■ **Tracing Facility - Turning it on.**

There are eight kinds of tracing that **nettl** can turn on.

Type of Tracing	Keyword	Mask Number
inbound header tracing	hdrin	0x80000000
outbound header tracing	hdrout	0x40000000
inbound PDU tracing	pduin	0x20000000
outbound PDU tracing	pduout	0x10000000
procedure entry/exit tracing	proc	0x08000000
state machine tracing	state	0x04000000
error tracing	error	0x02000000
log call tracing	logging	0x01000000

The command syntax is:

nettl	-traceon	<i>kind</i>	-entity	<i>subsystem</i>	OVEXTERNAL	-file	<i>file</i>
	-traceon	<i>kind</i>					
			-entity	<i>subsystem</i>			
	-file	<i>file</i>					

Specifies the kinds of tracing to be turned on for the specified subsystems.

Specifies the subsystems for which tracing is to be turned on. For tracing done by HP OpenView products, **OVEXTERNAL** must always be in the list.

Specifies the prefix of the trace file you want to create. Trace output will be written to *file*.TRC0 and *file*.TRC1. (See "How nettl Generates the Binary Data Files" for a description of these two trace files.)

Example 1:

Do error tracing for the **ovw** subsystem, putting the data in the trace files, **/tmp/nettl.TRC0** and **/tmp/nettl.TRC1**. As root, type the following command:

```
nettl -traceon error -entity OVW OVEXTERNAL -file /tmp/nettl
```

Example 2:

Do both inbound and outbound PDU tracing for the **ovw** subsystem, putting the data in the trace files, **/tmp/nettl.TRC0** and **/tmp/nettl.TRC1**. To enable more than one kind of tracing, either specify all the appropriate keywords, all the appropriate mask numbers, or the logical "OR" of the mask numbers. As root, type one of the following command variations:

```
nettl -traceon pduin pduout -entity OVW OVEXTERNAL -file /tmp/nettl
or
nettl -traceon 0x20000000 0x10000000 -entity OVW OVEXTERNAL -file /tmp/nettl
or
nettl -traceon 0x30000000 -entity OVW OVEXTERNAL -file /tmp/nettl
```

Note

Do not leave off the **-file** argument to **nettl -traceon**! If you do, unformatted trace output will be sent to your screen, almost certainly not what you wanted.

■ Tracing Facility - Turning it off.

While logging should always remain on, you should run the tracing facility only when a serious problem occurs and it is necessary to reconstruct in detail the sequence of events that caused the problem. After you have completed a trace, you should turn tracing off.

The command syntax is:

```
nettl -traceoff -entity subsystem
```

-traceoff	Specifies that tracing is to be turned off for the specified subsystems.
-entity subsystem	Specifies the subsystems for which tracing is to be turned off. To turn tracing off completely, specify all subsystems for which tracing is currently on.

Note that you cannot turn off trace kinds individually. To change the trace kinds enabled for a subsystem, turn tracing off for that subsystem, then turn tracing back on for the particular kinds in which you are interested.

Example 1:

Disable tracing for the **OVEXTERNAL** subsystem. As **root**, type the following command:

```
nettl -traceoff -entity OVEXTERNAL
```

■ Logging and Tracing Facilities - Turning both off.

To stop the logging/tracing facilities, as **root**, type the following command:

```
nettl -stop
```

How nettl Generates the Binary Data Files

The first time you run a **nettl** command for capturing logging data, the log file **/usr/adm/nettl.LOG00** is generated. Binary data captured from each successive **nettl** logging command is appended to this file, until it is filled. Then a second log file with the suffix **.LOG01** is created in which new data is stored. When this second file fills up, the **.LOG00** file is overwritten.

When you run a **nettl** command for capturing trace data, you specify a filename prefix, to which the suffix **.TRC0** is added. For each successive **nettl** tracing command, you can have the binary data appended to this file, or you can specify a new file in which to store the data. If you append binary data from successive traces to the same file, it will eventually fill up. A new file with the same prefix and the suffix **.TRC1** will then be created for storing new data. If this second file fills up, then a new trace file with the same prefix and the **.TRC0** suffix is created.

If you need to check the name of a log file before formatting it with **netfmt**, use a **nettl -status** command.

Formatting Network Data with netfmt

The command syntax is:

```
netfmt -f binary_file  
-f binary_file Specifies the log or trace file containing the binary data
```

It is possible to do elaborate filtering of the output, as **netfmt** has many other options that cannot be described here. Some of these options, for example, let you specify the output of a log or trace file to be filtered by timestamp, subsystem, or message class. For more information, see the **netfmt** man page.

Examples of netfmt Commands

The following example will help you start using the **netfmt** command.

```
To format the trace file, /usr/adm/nett1.TRC1. Type  
/etc/netfmt -f /usr/adm/nett1.TRC1
```

The next section shows what the output of **netfmt** looks like.

Log and Trace File Output

Below are two complete examples of using the **nett1** and **netfmt** commands, and the final output you might see.

Example 1:

1. You ran a **nett1** logging command for informative messages in the **ovs** subsystem, in order to observe the startup of the **ovspmd** process, by typing:

```
nett1 -log i e d -entity OVS OVEXTERNAL
```

The binary data was stored in the **/usr/OV/log/nett1.LOG00** log file.

2. You formatted and displayed the binary log file by typing:

```
netfmt -f /usr/OV/log/nett1.LOG00 | more
```

The output would be similar to the following:

*****OpenView*****@#%

Timestamp : Wed Nov 17 1993 17:28:58.441517
Process ID : 22756 Subsystem : OVS
User ID (UID) : 0 Log Class : INFORMATIVE
Device ID : -1 Path ID : -1
Connection ID : -1 Log Instance : 0

Software : ovspmd
Hostname : pinetum.cnd.hp.com

Initialized successfully:
ovspmd

*****OpenView*****@#%

Timestamp : Wed Nov 17 1993 17:28:59.352086
Process ID : 22756 Subsystem : OVS
User ID (UID) : 0 Log Class : INFORMATIVE
Device ID : -1 Path ID : -1
Connection ID : -1 Log Instance : 0

Software : ovspmd
Hostname : pinetum.cnd.hp.com

Accepted request connection (fd 5)

*****OpenView*****@#%

Timestamp : Wed Nov 17 1993 17:28:59.355761
Process ID : 22756 Subsystem : OVS
User ID (UID) : 0 Log Class : INFORMATIVE
Device ID : -1 Path ID : -1
Connection ID : -1 Log Instance : 0

Software : ovspmd
Hostname : pinetum.cnd.hp.com

Received request, fd 5
[9] [START] ""

*****OpenView*****@#%

Timestamp : Wed Nov 17 1993 17:28:59.358398
Process ID : 22756 Subsystem : OVS
User ID (UID): 0 Log Class : INFORMATIVE
Device ID : -1 Path ID : -1
Connection ID : -1 Log Instance : 0

Software : ovspmd
Hostname : pinetum.cnd.hp.com

Building master M table from /usr/OV/conf/ovsuf

*****OpenView*****@#%

Timestamp : Wed Nov 17 1993 17:28:59.366810
Process ID : 22756 Subsystem : OVS
User ID (UID): 0 Log Class : INFORMATIVE
Device ID : -1 Path ID : -1
Connection ID : -1 Log Instance : 0

Software : ovspmd
Hostname : pinetum.cnd.hp.com

Reporting M status, code = SUCCESS

object manager name: OVsPMD
behavior: OVs_PMD
state: RUNNING
PID: 22756
exit status: -

Example 2:

1. You ran a `nettl` tracing command for procedure tracing in the `OVEXTERNAL` subsystem, in order to observe some procedure calls in `ipmap`, by typing:

```
nettl -tracoon proc -entity OVEXTERNAL -file /usr/adm/nettl
```

The binary data was stored in the `/usr/adm/nettl.TRC0` trace file.

2. You formatted the binary trace file by typing:

```
netfmt -f /usr/adm/nettl.TRC0
```

A brief fragment of the output appears below:

```
*****OpenView*****@#%
```

```
Timestamp      : Wed Nov 17 1993 18048:44.443027
Process ID     : 23059             Subsystem      : OVW
User ID ( UID ) : 214             Trace Kind     : Proc. Entry/Exit
Device ID      : -1              Path ID       : -1
Connection ID  : -1
```

```
Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
```

```
-----
FieldBinding::FieldBinding(OVwFieldBinding) entered
```

```
*****OpenView*****@#%
```

```
Timestamp      : Wed Nov 17 1993 18:04:44.444646
Process ID     : 23059             Subsystem      : OVW
User ID ( UID ) : 214             Trace Kind     : Proc. Entry/Exit
Device ID      : -1              Path ID       : -1
Connection ID  : -1
```

```
Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
```

```
-----
FieldBinding::init() entered
```

```

*****OpenView*****@##%
Timestamp      : Wed Nov 17 1993  18:04:44.446053
Process ID     : 23059              Subsystem      : OVW
User ID ( UID ) : 214              Trace Kind     : Proc. Entry/Exit
Device ID      : -1                Path ID        : -1
Connection ID  : -1

Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
-----
FieldBinding::init() exit

*****OpenView*****@##%
Timestamp      : Wed Nov 17 1993  18:04:44.447404
Process ID     : 23059              Subsystem      : OVW
User ID ( UID ) : 214              Trace Kind     : Proc. Entry/Exit
Device ID      : -1                Path ID        : -1
Connection ID  : -1

Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
-----
FieldBinding::copyFieldValue() entered

*****OpenView*****@##%
Timestamp      : Wed Nov 17 1993  18:04:44.448754
Process ID     : 23059              Subsystem      : OVW
User ID ( UID ) : 214              Trace Kind     : Proc. Entry/Exit
Device ID      : -1                Path ID        : -1
Connection ID  : -1

Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
-----
FieldBinding::copyFieldValue() exit

*****OpenView*****@##%
Timestamp      : Wed Nov 17 1993  18:04:44.450233
Process ID     : 23059              Subsystem      : OVW
User ID ( UID ) : 214              Trace Kind     : Proc. Entry/Exit
Device ID      : -1                Path ID        : -1
Connection ID  : -1

Software       : /usr/OV/bin/ipmap
Hostname       : pinetum.cnd.hp.com
-----
FieldBinding::FieldBinding(OVwFieldBinding) exit

```

Characterizing the Problem

Symptoms are visible circumstances that indicate a problem. Whenever you encounter a symptom, collect the basic information that follows.

Scope: What is Affected?

Distinguish Local vs. Remote Problem

If you suspect a problem on a remote node or agent system, try to duplicate it on a local node or manager system, and vice-versa. If you cannot duplicate the problem on the local node, the problem is probably associated with the remote node. If you can duplicate the problem on the local node, the problem may be in the manager system, somewhere in the link, or may be a problem common to the agents.

Remote Nodes' Level of Manageability

What type of remote node are you trying to communicate with when the problem occurs?

The amount of information that the HP OpenView network management system can collect depends on whether a remote node

- supports SNMP or not (proxied)
- supports a partial, full, or extended (non-standard) MIB
- is an HP node or not
- is a TCP/IP node or not

The manager gets the most information from an HP SNMP node, less information from other SNMP nodes, least information from non-SNMP, TCP/IP nodes, and no information from non-SNMP, non-TCP/IP nodes. For more details, see “Levels of Support” in the *Installation Guide*.

Affected Parts of the HP OpenView Network Management System

What part of the HP OpenView network management system is affected? All operations, or just some?

Context: What Changed?

Determine what may have changed on your network or product configuration: hardware, software, files, security, utilization, etc.

Also determine what is unique about a particular node that is having problems.

Duration: How Long or How Often?

Is the problem consistent (fails every time) or intermittent (fails sometimes)?

Context: What Action Was Performed?

When the problem occurred, what was happening? For instance,

- What operation was selected?
- What command was executed?
- What mouse operation was performed?
- What data was requested or sent?
- Was an event logged in the `/usr/ov/log/trapd.log` file?
- Was an error or disaster logged through `nettl`?
- Does `ovstatus` show all daemons running?



General Product Troubleshooting Considerations

Consider the following items when troubleshooting the manager product.

- The HP OpenView network management system is memory and swap-space intensive. Consider that a problem may be due to resource exhaustion when troubleshooting this product.
- SNMP is based on User Datagram Protocol, or UDP, which is an unreliable protocol (no error checking and no guarantee of message receipt). This may cause occasional failures of manager-agent communication.
- If you are using the Berkeley Internet Name Domain (BIND) Service on your network, each `gethostbyaddr` request for the IP address of a host with multiple IP addresses could return a different hostname.

When You Need More Information

Other information that may assist your troubleshooting of the HP OpenView product is available in

- 
- the release note files in the `/usr/OV/ReleaseNotes` directory
 - the HP OpenView Windows' online `help` operation
 - the operational overview in chapter 1, "Introduction and Operational Overview"
 - the "Reference" appendix
 - the Request for Comment (RFC) documents referenced in chapter 1, "Introduction and Operational Overview"
 - manuals for other Hewlett-Packard managers, for example, the *HP OpenView Interconnect Manager Installation and User's Guide*

If a problem does not appear to be with the network management products themselves, see your

- networking documentation for network troubleshooting procedures
 - system documentation for system troubleshooting procedures
 - X Windows documentation for X Windows troubleshooting procedures
- 

Troubleshooting Activation Key Problems

If you have problems with the activation key, refer to the troubleshooting information in the *Installation Guide*.

Troubleshooting IP Discovery and Layout

This section explains

- How IP discovery and layout work.
- Why IP discovery and layout may not discover all the nodes on your network.
- What to do if HP OpenView network management system does not discover any nodes on your network or if your map is drawn incorrectly.

How IP Discovery and Layout Work

netmon, the background process that discovers the nodes on your network, uses SNMP requests to find out about the nodes on your network.

To discover the nodes on the network, **netmon** needs access to the following information:

- The subnet mask from the agent on the management station.
- The address of the default router in the management station's routing table.
- SNMP information from, at the minimum, the default router, and from other routers and nodes on the network.

For **netmon** to work, it requires the following:

- The management station must be correctly configured for networking.
- The management station must be running an SNMP agent.
- Nodes must be up and responding to **ping** requests to be discovered.
- All gateways/routers and the management station must have correctly configured subnet masks for all interfaces.

netmon works best under these situations:

- The more routers in the network, the better; and those routers should be running SNMP agents.
- The more nodes running SNMP, the better.
- The more SNMP information **netmon** gets the better, for example, valid ARP Caches.

Information about the discovered nodes are stored in the topology database and is used to automatically generate the network map.

Over time, the HP OpenView network management system will discover all new nodes on the network. However, if a new node never talks with a gateway or other nodes on the network that supports SNMP, the manager may not find it. In these instances, you can **ping** that node from the management station and the manager will discover the node; or you can add the node manually. `/usr/ov/log/trapd.log` shows each node as it is discovered; you can see when discovery gets “caught up.”

Before you start to troubleshoot **netmon**, allow the process sufficient time to discover your network. The amount of time required by **netmon** to discover your network depends on how many nodes you have on your network, what type of system you have, and whether or not the nodes support SNMP. Because of these variables, the amount of time required will vary.

EXAMPLES:

Let's say you have about 800 to 900 nodes in your network and your gateways support SNMP.

- If you run **netmon** on an HP-UX Series 700 system, **netmon** will discover the first 500 nodes in about one hour. During the next hour, **netmon** will discover about 400 nodes. So in this case, wait at least two hours before you start to troubleshoot. Note that the more nodes that have been discovered, the slower **netmon** will discover new nodes. The reason for this is that **netmon** requires time to monitor the nodes that it already knows about.
- If you run **netmon** on an HP-UX Series 400 system, **netmon** will discover about 300 nodes per hour. So in this case, wait at least three hours before you start to troubleshoot.

For more information about the performance of the HP OpenView network management system, refer to the *HP OpenView SNMP Platform Performance and Configuration Guide, with HP OpenView Network Node Manager Examples*. This information can also be found in the `/usr/ov/ReleaseNotes` directory.

Why IP Discovery May Not Discover All Nodes

The **netmon** discovery process discussed in the previous section may not discover all the nodes on the network under the following conditions:

- If the SNMP community name configured on the management station does not match the community name configured on the nodes on the network, the discovery process will be hindered.
- If the network management station's default router does not support SNMP or if the network has few SNMP nodes, the discovery process may not discover all nodes.

- If the nodes on the network do not talk to each other or if the nodes only talk among themselves in an isolated area that is connected to the network through a gateway that does not support SNMP, the discovery process may fail.
- If the network uses bridges and not routers, the discovery process may not discover all nodes because bridges do not supply any information about what nodes are on the network.
- Nodes that do not support IP cannot be discovered.

What to Do if IP Discovery Fails

This section has two checklists.

- The first checklist has general suggestions for what to do if the HP OpenView network management system does not discover the nodes on your network correctly.
- The second checklist shows some common symptoms and gives suggestions for how to solve these problems.

General Suggestions

- Make sure that the nodes are running and that the nodes are responding to **pings** and SNMP requests. Issue the **ping** commands and the SNMP requests (**snmpget** and **snmpwalk**) from the command prompt.
- Make sure the routing tables are configured properly. As discussed in the section “How IP Discovery and Layout Work,” **netmon** uses the default router in the routing table to start the discovery process.
- Make sure that the information (such as IP addressing) configured for, or on, the map object itself is accurate. To verify this information, use the map's **Monitor: Description** operation.
- Make sure the subnet masks are correct. If the subnet masks on the management station, gateways/routers, or nodes in the seed file are wrong, discovery will fail.

Symptoms of incorrect subnet masks may include

- **netmon** may discover nodes from the internet that you do not know about.
- Nodes do not show up on the map.
- Incorrect networks are discovered.

To determine what subnet masks the agent returns to the HP OpenView network management system, do one of the following:

- Select the management station and use the **Monitor: Description** operation.
- Select the management station and query the **.iso.org.dod.internet.mgmt.mib-2.ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask** MIB object using either the **Monitor: MIB Values -> Browse MIB** operation or execute the **snmpget** or **snmpwalk** command directly from the command prompt.

- Use the tools provided by your system. On an HP-UX, you can use the `/etc/ifconfig interface` command, where **interface** is usually `lan0`, `lan1`, and so forth. `ifconfig` tells you how the system is actually configured; it may not be what the agent returns. For information, see your system documentation.

Alternatively, if the subnet masks on the management system is correct, use the `ovttopodump` command to show you what subnet masks are being used for each network.

To solve problems with the map caused by incorrect subnet masks, follow these steps:

- a. Correct the problem with the subnet masks on the networks object that have the problem.
- b. Delete any networks that may have incorrect subnet masks in the map. Use the **Monitor: Description** operation to see the subnet masks.

Either add the networks again or let them be rediscovered.

Note

Non-contiguous subnet masks are not supported.

Also, not supported is when in hierarchical subnets, two subnets have the same IP subnet address, but different subnet masks.

- Make sure the community name is configured correctly. If the community name on the default router or the community name on nodes in the seed file is wrong or does not match, `netmon` will not be able to get the information that it needs to discover the nodes on the network. The community name on a node's agent must match the community name configured on the manager.

Symptoms of incorrect community names include

- The default router is not discovered.
- The default router is shown as a host on the map.
- Error messages appear, such as "timeout" or "no such name."

To verify that the community names match, select the node and use the

Monitor: MIB Values -> Browse MIB operation to verify the community name. For example, Cisco routers have the ability to change the community name. The default community name is **public**. If you change the community name on the agent, you also *must* configure the community name for that agent on the manager. To configure the community name on the manager, use the **Options: SNMP Configuration** operation. You must also configure community names before you can do SNMP SetRequests.

- Make sure that the SNMP agent on the node is running.

Common Symptoms

- If the HP OpenView network management system does not discover any nodes when run for the first time, check the following:
 - Make sure that the SNMP agents are installed and running on the agent systems, particularly on gateways. The problem may be that there are no SNMP agents installed. If you have very few SNMP nodes on your network, try to run **loadhosts** (see the **loadhosts** man page), or try to run **netmon** with the “-J” option. See the **netmon** man page.
 - Make sure agents with non-default community names are added to the **Options: SNMP Configuration** operation on the manager system.
 - Make sure that the **snmpd** agent is installed and running on the manager system. An **snmpd** agent must be running on the manager system for discovery to function correctly.
 - If you specified a seed file, make sure that the seed file exists and that you are using the correct name. The problem may be that the specified seed file on **netmon**'s command line does not exist. Also, make sure that you use all the nodes that you want to use as seeds in the seed file and that all the nodes listed in the seed file support SNMP. Do not use bridges and hubs as seeds.
 - Make sure you can **ping** your default router and the closest SNMP supporting router.
 - Make sure your default route is configured correctly. To find the default route, use **netstat -r**.
- If strange networks appear on the map, such as a network having a Class A, Class B, or Class C network number when you expect a subnetted network number, ensure that the nodes appearing on those networks have correctly configured network (subnet) masks. Incorrect subnet masks can cause the addition of incorrect networks on the map. **netmon** may discover and add nodes from outside of your administrative domain to an un-subnetted or incorrectly subnetted network. For networks to be drawn correctly, subnet masks must be correctly configured on all IP interfaces on the following:
 - All gateways
 - The manager system
 - Any nodes in a seed file for **netmon**

Incorrect network (subnet) masks on other nodes can generate many **Inconsistent subnet mask** events. The map's **Monitor: Description** operation displays the node's configuration. Make sure the subnet mask is correct.

To solve problems with the map caused by incorrect subnet masks, follow these steps:

- a. Correct the problem with the subnet masks on the network object that has the problem.
- b. Delete any networks that may have incorrect subnet masks in the map. Use the **Monitor: Description** operation to see the subnet masks.

Either add the networks again or let them be rediscovered.

Note

Non-contiguous subnet masks are not supported.

- If isolated information in the map database is inaccurate, do the following:
 - Use the graphical network map's **Edit** operations to correct the inaccuracy.
 - Use the **Diagnose: Network Connectivity -> Demand Poll** operation to update the database entries for the nodes or verify monitoring of the nodes.
- If all or most of the map database is inaccurate, do the following:
 - Try shortening polling intervals for better monitoring accuracy (though this generates more polling traffic) using the **Options: Topology/Status Polling: IP** operation.
 - Use the **Monitor: Description -> Selected Objects** menu item to verify the subnet masks for the subnets.
 - If routers are incorrectly configured, correct the configuration and restart IP discovery.
- If routers are not showing up on your network, check to see if the routers are “backup” routers, that is, routers used in an emergency when another router goes down. **netmon** may not discover backup routers, because **netmon** looks only for nodes on the network that are actually used. Therefore, a backup router will be discovered either as a host or it may not be discovered at all. To solve this problem, follow these steps.
 - Manually add the router to the map. For information about how to manually add the router, see the *HP OpenView Network Node Manager User's Guide*.
 - **ping** the node to determine the initial status condition and turn the node green.
 - Use the **Diagnose: Network Connectivity -> Demand Poll** to get the information that **netmon** needs to connect the router in the proper place on the map.
- If the HP OpenView network management system incorrectly identifies an SNMP node as a non-SNMP node, the problem may be that all the **netmon** SNMP requests to a node fail. During SNMP requests, **netmon** asks for system information, interface table, IP address table, and other information in a specific order. **netmon** does four types of queries: an initial configuration check, which is done at initial discovery, a **Demand Poll**, new discovery polls, and automatic daily configuration checking. If the node does not reply with a system object id, **netmon** assumes that the node does not support SNMP. If, however, any one of these three requests succeeds, **netmon** assumes the node *does* support SNMP. Note that the SNMP requests can fail if the transport is very busy. SNMP runs over UDP.

The most common reason that **netmon** incorrectly identifies a node that supports SNMP as not supporting SNMP, is that the community name is not configured correctly. If the community name is not configured or if the manager uses the wrong community name in the request, the agent does not respond. The agent does not return a message indicating wrong community name. Another common reason for **Demand Poll** failures is that the agent returns incorrect information.

To verify that you have a problem with the agent, use the

Monitor: MIB Values -> Browse MIB operation. It may also be helpful to issue the **snmpwalk** command on the following tables:

- interface table (.iso.org.dod.internet.mgmt.mib-2.interfaces)
- IP address table (.iso.org.dod.internet.mgmt.mib-2.ip.ipAddrTable)
- ARP Cache table (.iso.org.dod.internet.mgmt.mib-2.ip.ipNetToMediaTable)

Look for the following

- Does the information that is being returned make sense?
- Do the entries in the interface table have a corresponding entry in the IP address table?

If you have verified that the problem is with the agent, send the information to the vendor of the agent.

- If the default router shows up as a host on the map, verify the following:
 - Make sure the community name on the agent and the manager match.
 - Make sure that IP forwarding is set to "forwarding" *and* that the router has more than one interface card. You can use the **Monitor: MIB Values -> Browse MIB** operation to see what values are returned by the particular routes.
- If you have a host that shows up as a router, it may be because the host has multiple interfaces. To solve this problem, the node must have IP forwarding turned off. Then, delete the node from the map and then manually add the node as a "host."
- If the HP OpenView network management system does not discover secondary addresses.

Secondary addresses are logical interfaces that are configured in addition to the physical interfaces on a router. Typically, this is done on Cisco devices but other vendor's routers may also allow this functionality. Unfortunately, these additional interfaces are not included in the IP address table returned through SNMP. Therefore, the HP OpenView network management system is unaware that these interfaces exist.

When an ip address is found in an ARP Cache and it resolves to the same official name as a router, the IP address is added as an additional interface. When the router's node submap is displayed, the primary interfaces (the ones returned through SNMP) have labels that contain the first word of the interface description. The secondary addresses (the ones discovered in an ARP Cache) have only their IP addresses as their labels.

The HP OpenView network management system can discover these interfaces if and only if:

- The secondary address resolves to the same official name as the router.
- The secondary address is connected to an existing subnet.

If the HP OpenView network management system fails to discover all the configured secondary addresses, they can be manually added as interface symbols in the router's node submap. If the interface is connected to a new subnet, that subnet must also be manually added. The HP OpenView network management system will not automatically add the subnet symbol.

The secondary addresses can be placed in a seed file that will be used by the **netmon** process for discovery (see the man page for **netmon**). In this case, the secondary address interface symbols and the corresponding networks will be added to the map automatically.

Troubleshooting Background Processes

This section explains the background processes of the HP OpenView network management system and suggests actions to take if you have problems with one of the following background processes. The background processes are:

- **netmon**
- **ovtopmd**
- **ovwdb**
- **snmpCollect**
- **trapd**
- **ovactiond**

In addition to explaining the background processes, this section explains how to troubleshoot **ovspmd**, the process that monitors and controls the background processes.

The background processes and **ovspmd**, should always be running whenever the HP OpenView network management system is running. Normally, the background processes are started through **ovstart**. To verify that they are running, enter the following:

ovstatus

The result should look similar to the following:

```
object manager name: OVSPMD
behavior:            OVs_PMD
state:               RUNNING
PID:                 3509
exit status:         -

object manager name: trapd
behavior:            OVs_WELL_BEHAVED
state:               RUNNING
PID:                 3512
last message:        Initialization complete.
exit status:         -

object manager name: ovwdb
behavior:            OVs_WELL_BEHAVED
state:               RUNNING
PID:                 3511
exit status:         -
```

object manager name: ovtopmd
behavior: OVS_WELL_BEHAVED
state: RUNNING
PID: 4119
last message: Initialization complete.
exit status: -

object manager name: netmon
behavior: OVS_WELL_BEHAVED
state: RUNNING
PID: 4120
last message: netmon started
exit status: -

object manager name: snmpCollect
behavior: OVS_WELL_BEHAVED
state: RUNNING
PID: 4121
last message: Initialization complete.
exit status: -

object manager name: ovactiond
behavior: OVS_WELL_BEHAVED
state: RUNNING
PID: 4122
last message: Initialization complete.
exit status: -

If **ovstatus** indicates that an object manager was not found, the problem may be caused by one of the following:

- The configuration scripts for **SNMPRUN** or **NNMGR** were not run. If the configuration scripts have not been run, execute:
ovconfigure -p SNMPRUN
ovconfigure -p NNMGR
- The **ovspmd** configuration is corrupted. If the **ovspmd** configuration is corrupted, see the “The ovspmd Background Process” section later in this chapter.
- The executable file has been removed or moved from its expected location. **ovsuf** contains the pathname to each executable. Make sure they exist where they are supposed to be.

For problems with individual background processes, see the appropriate section on how to troubleshoot a specific background process. For example, if the problem is with **ovtopmd**, see “The ovtopmd Background Process” section.

Some of the background processes depend on other processes to operate properly. Table 4-3 shows the dependencies.

Table 4-3. Background Process Dependencies

Background Process	Dependencies
netmon	ovwdb, trapd, ovttopmd, and snmpd
ovtopmd	ovwdb and trapd
ovwdb	no dependencies
snmpCollect	ovwdb, trapd, and ovttopmd
trapd	no dependencies
ovactiond	trapd

The ovspmd Process

The **ovspmd** process monitors and controls the HP OpenView background processes. These background processes are supposed to run as children of **ovspmd**. The **ovspmd** process should be running whenever the background processes are running. Normally, **ovspmd** is started by **ovstart**.

To verify the status of the processes, type:

ovstatus

If **ovstatus** provides output indicating that all processes are running, the problem is not **ovspmd**. If you do not receive the expected output, either **ovspmd** has a communication problem or the **ovspmd** process died. Continue to "Determining ovspmd Problems."

Determining ovspmd Problems

You need to determine whether the problem is related to **ovspmd** or to a background process. If you determine you have an **ovspmd** problem, you need to determine what type of problem it is. Here are some indications of **ovspmd** problems:

- If the **ovstatus** output indicates that one of the background processes is not running, type **ovstart**

This should start any process that is not running. If any process fails to start, an error message will appear, indicating which process is not running. If this happens, type **ps -ef**. This will list currently running processes.

Do *one* of the following:

- If the output indicates a certain process is running, but **ovspsmd** says the process is not running (using **ovstatus** or **ovstart**), **ovspsmd** does not know that process is running.

(This problem can occur when **ovspsmd** is shut down with the **kill -9** command.)

- If **ovstatus** output are in agreement that a certain process is not running, that process has a problem. See the troubleshooting section for that process.

When you run **ovstart**, the error message you receive about the process should provide information that will help resolve the problem.

- If you receive *one* of the following error messages

ovstart: ovspsmd is not running. It may have failed to start.

ovstart: unable to contact ovspsmd: error message

you need to verify that the **ovspsmd** sockets directory exists. Go to “Recreating the **ovspsmd** Sockets Directory.”

- If you receive an **ovspsmd** error message in the following format

ovspsmd: /usr/OV/conf/ovsuf: error message

you may have a corrupt **ovsuf** configuration file. Go to “Recreating the **ovsuf** Configuration File.”

Recreating the **ovspsmd** Sockets Directory

Use this procedure when **ovstart** is unable to start or connect to the **ovspsmd** process.

1. Verify that the **/usr/OV/sockets** directory still exists. **ovspsmd** creates the **OVsPMD_MGMT** and **OVsPMD_REQ** socket files in this directory, which enable **ovspsmd** to communicate with the other background processes. If this directory no longer exists, recreate it by typing

```
mkdir /usr/OV/sockets
```

2. Find out if any of the background processes are running by typing

```
ps -ef
```

3. Kill all the HP OpenView background processes that are running using the **kill** command. *Do not* use **kill -9**. For a list of the processes, see “Troubleshooting Background Processes” earlier in this chapter.

4. To restart the processes and verify that they are starting properly, type **ovstart -v**

Recreating the ovsuf Configuration File

Use this procedure when you receive an **ovspmd** error message.

1. Verify that the **ovsuf** configuration file is corrupted. If **ovsuf** is corrupted, you will get a listing of the corrupted lines when you run **ovstart**. If **ovsuf** is corrupted, continue to step 2.

2. Stop any background processes that may be running by typing

```
ovstop
```

3. Empty the **/usr/OV/conf/ovsuf** file by typing

```
cat /dev/null > /usr/OV/conf/ovsuf
```

4. Recreate the **/usr/OV/conf/ovsuf** file by typing

```
ovconfigure -p SNMPRUN  
ovconfigure -p NNMGR
```

If you are running a configuration other than the standard configuration, see “Modifying the Start-Up Configuration” in chapter 3.

5. To restart the processes and verify that they are starting properly, type

```
ovstart -v
```

If **ovstart** still fails, you may have another **ovspmd** problem. Go back to “Determining ovspmd Problems.”

The netmon Background Process

The **netmon** process uses various ICMP requests (**ping**) to poll nodes for status, and SNMP requests to poll nodes for MIB values. When **netmon** detects a change, the process updates the topology database (through **ovtopmd**) with the change and sends the appropriate event to **trapd**. **trapd** then logs the event to **/usr/OV/log/trapd.log** and forwards it to **ipmap**, **ovactiond**, and **xnmevents**. **ipmap** gets the information from **ovtopmd** and updates the HP OpenView Windows (**ovw**) map and the Map database. **xnmevents** updates its **Events** notification operations accordingly. Figure 4-1 illustrates these **netmon** interactions.

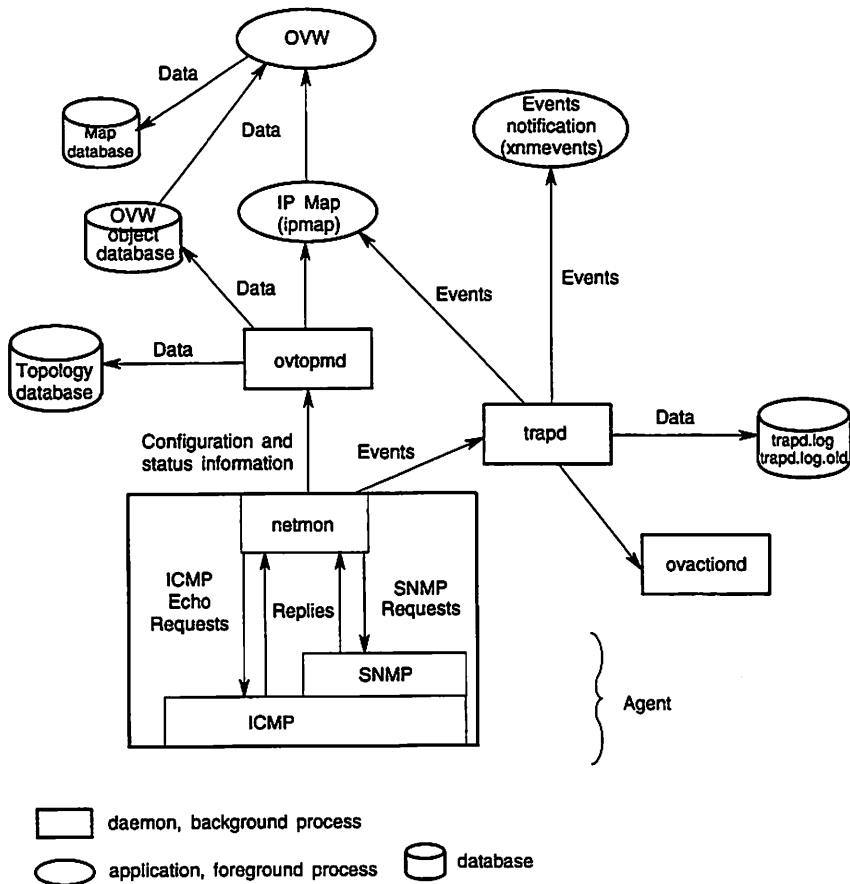


Figure 4-1 netmon Interactions

You can control **netmon**'s polling through the map's **Options: SNMP Configuration** and **Options: Topology/Status Polling** -> IP operations. The **SNMP Configuration** operation modifies the `/usr/OV/conf/ovsnmp.conf_db` database. The **Topology/Status Polling** operation stores **netmon**'s polling intervals in the `/usr/OV/databases/openview/topo/polling` file. Do not modify this file directly; it contains order-sensitive data.

The **xmmevents** process checks every five minutes to see if **netmon** and **snmpCollect** are running. If either process is not running, **xmmevents** will display a dialog box and report the inactive process. (For information on changing the five-minute default polling interval or on disabling this polling feature, see the `/usr/OV/app-defaults/XNm` file.

Polling is scheduled on a per node basis, but is not deterministic. Therefore, polling of a node may not occur when scheduled if **netmon** gets behind due to its polling load for a particular time interval. Also, polling of manually added nodes and nodes changed from unmanaged to managed may not occur immediately after they are added or managed.

While you are unmanaging a network or segment during active discovery, a few managed nodes may appear on that network. To avoid this problem, do one of the following:

- Wait until the program finishes active discovery of the map.
- Turn polling off while unmanaging the network.

Only one instance of **netmon** may run on the manager system; the `/usr/OV/databases/openview/topo/netmon.lock` file enforces this restriction.

The following tools can help you troubleshoot **netmon** process problems.

- Use the **Diagnose: Network Connectivity** -> **Demand Poll** operation to test or cause immediate **netmon** polling activity to a node.
- Check the `/usr/OV/log/trapd.log` file or the **Error Events Browser** window for **netmon** process errors.
- Use **netmon** tracing to help isolate problems. For information see the section “Logging and Tracing” earlier in this chapter, or see the **netmon** man page.

The ovtopmd Background Process

The **ovtopmd** process is responsible for maintaining the IP topology database that is used by **netmon** and **ipmap**. It provides both enforcement of rules for correctness of the IP topology, and caching of the data to improve the performance of the system.

ovtopmd should be running whenever the HP OpenView network management system is running. Normally it is started through **ovstart (1M)**. To verify that it is up and running, type

```
ovstatus ovtopmd
```

You should see something like this:

```
object manager name: ovtopmd
behavior:           Ovs_WELL_BEHAVED
state:              RUNNING
PID:                4119
last message:       Initialization complete.
exit status:        -
```

If **ovstatus** gives an error similar to this:

```
ovspmd: /usr/OV/conf/ovsuf: object manager ovtopmd not found
```

the problem may be one of two things.

- The configuration scripts for the **OVIIPMAP** fileset were not run. If the configuration scripts have not been run, type

```
ovconfigure -p SNMPRUN
```

- The **ovspmd** configuration is corrupted. If the **ovspmd** configuration is corrupted, see the “The ovspmd Process” section earlier in this chapter.

If **ovstatus** knows about **ovtopmd**, but indicates that it is not running, enter the following:

```
ovstart -v
```

Normally, this should start **ovtopmd** and all daemons that depend on it, as well as all the daemons on which **ovtopmd** depends. In particular, **ovtopmd** depends on **ovwdb** being started. If **ovwdb** does not start, **ovtopmd** will not be able to execute successfully. In addition to **ovwdb** executing, **ovtopmd** relies on the field definitions being loaded into **ovwdb**, which normally happens through the configuration scripts for the **OVIPMAP** fileset.

If **ovtopmd** gives an error indicating that it was unable to map a field name into a field ID, do the following:

- Verify that **\$LANG** is set correctly and that the **/usr/OV/fields/\$LANG** directory exists and contains at least the following files: **ovw_fields**, **ip_fields**, **snmp_fields**, and **topm_fields**. The **\$LANG** environment variable provides support for localizing native language. If not otherwise defined, **\$LANG** is assumed to be **C**, which means that no native language support is provided. (For more information on native language localization, see the manual page for **lang(5)**.)
- Type

```
ovconfigure -p SNMPRUN
```

This should load any necessary field definitions into **ovwdb**, and then start **ovtopmd** and **netmon**.

By default, **ovtopmd** uses a TCP socket bound to port 8888. If **ovtopmd** indicates an error attempting to bind to this port number because another application is already using it, follow these steps:

1. Make sure that you do not have another **ovtopmd** daemon using port 8888.

The most common reason for errors binding to the TCP socket is the presence of another **ovtopmd** daemon. If an **ovtopmd** daemon was started outside of **ovstart**, the **ovstop**, **ovstart**, and **ovstatus** commands will not recognize this daemon. To prevent this problem, always start all HP OpenView daemons with the **ovstart** command and stop them with the **ovstop** command.

If another application is using port 8888, continue with the next step.

2. Change the port number to any unused port by changing the entry for **ovtopmd** in **/etc/services**:

```
ovtopmd 8888/tcp          # OpenView IP Topology daemon
```

Choose a number other than 8888 that does not conflict with other services listed in **/etc/services**.

3. Reconfigure the HP OpenView network management system by typing

```
ovconfigure -p SNMPRUN
```

4. If **ovtopmd** still fails to run successfully from **ovstart**, contact your support representative.

Note

If you are using the Network Information Service (NIS), make sure you update your NIS Master Server.

Once **ovtopmd** has been started through **ovstart**, you can verify that it is responding to requests by executing **/usr/OV/bin/ovtopodump**. By default, **ovtopodump** will print a single line of output for each object in the topology. You can also get verbose listings and summary information. For more information see the manual page for **ovtopodump**. For example, if you type

```
ovtopodump -l
```

your result will look something like

```
TOPO OBJECT ID: 70
TIME CREATED: Fri Apr 10 14:52:02 1993
TIME MODIFIED: Fri Apr 10 14:52:02 1993
GLOBAL FLAGS:
NUMBER OF NETWORKS: 13
NUMBER OF SEGMENTS: 19
NUMBER OF NODES: 238
NUMBER OF INTERFACES: 281
NUMBER OF GATEWAYS: 21
```

This prints the summary information for the entire IP topology. If **ovtopodump** fails, this indicates some problem with **ovtopmd**. Try stopping all daemons with **ovstop**, and then restarting them with **ovstart**.

If the **ovstart** succeeds, but **ovtopodump** still fails, verify that at least one of the following is true:

- The IP address of the loopback interface is **127.0.0.1**.
- There is an entry for "localhost" that has the correct address for the loopback interface.
- The IP address for the localhost's hostname is correct. (That is, do a **hostname**, and then look up that name with **nslookup**, and verify that the address returned is correct.)

In an HP-UX clustered environment, **ovtopmd** will execute only on the root server, but you should be able to run **ovtopodump** from any cluster node that is of the same architecture as the server. In this case, for correct operation the cluster node must be able to determine the correct IP address of the cluster root server, as based on the root-server's name as entered in **/etc/clusterconf**.

If **ovtopodump** still fails, contact your support representative.

In addition, **ovtopmd** maintains consistency with the HP OpenView Windows object database maintained by **ovwdb**. You can use the **ovtopofix** command to do the following consistency checks and updates:

- Check object existence in both the topology and **ovwdb** databases.
- Ensure that the managed state is consistent.

- Remove incomplete interface objects and incomplete nodes created by the discovery process from the HP OpenView Windows object database.
- Update the presentation time to force **ipmap** to update the objects' status during map synchronization.
- Update the creation time of the object to force **ipmap** to add the object during map synchronization.

To run **ovtopofix** and check for problems type

```
ovtopofix -n
```

To run **ovtopofix** and fix any problems found, follow these steps:

1. Exit **ovw** using the **File: Exit** menu item.
2. Stop **netmon** by typing **ovstop netmon**.
3. Type **ovtopofix**.
4. Restart **netmon** by typing **ovstart netmon**.

For more information see the **ovtopofix(1M)** man page.

The ovwdb Background Process

The **ovwdb** process controls the HP OpenView Windows object database. Here are some problems that can be caused by **ovwdb** and possible solutions to the problems.

ovwdb should be running whenever the HP OpenView network management system is running. Normally it is started through **ovstart (1M)**. To verify that it is up and running type

```
ovstatus ovwdb
```

You should see something like this

```
object manager name: ovwdb
behavior:           Ovs_WELL_BEHAVED
state:              RUNNING
PID:                4115
last message:       Initialization complete.
exit status:        -
```

If **ovwdb** will not start, make sure you have an activation key on the system. To obtain and install the activation key, refer to the *Installation Guide*. Then run

```
ovconfigure -p SNMPRUN
```

By default, **ovwdb** uses a TCP socket bound to port of 9999. If **ovwdb** indicates an error attempting to bind to this port number because another application is already using it, follow these steps:

1. Make sure that you do not have another **ovwdb** daemon using port 9999.

The most common reason for errors binding to the TCP socket is the presence of another **ovwdb** daemon. If an **ovwdb** daemon was started outside of **ovstart**, the **ovstop**, **ovstart**, and **ovstatus** commands will not recognize this daemon. To prevent this problem, always start all HP OpenView daemons with the **ovstart** command and stop them with the **ovstop** command.

If another application is using port 9999, continue with the next step.

2. Change the port number to any unused port by changing the entry for **ovwdb** in **/etc/services**.

```
ovwdb    9999/tcp          # OpenView Object Database daemon
```

3. Reconfigure the HP OpenView network management system by typing

```
ovconfigure -p SNMPRUN
```

4. If **ovwdb** still fails to run successfully from **ovstart**, contact your support representative.

Note

If you are using the Network Information Service (NIS), make sure you update your NIS Master Server.

If you notice a great deal of swapping and the system performance does not meet your expectations, you can do one of two things:

- Edit the **/usr/OV/1rf/ovwdb.1rf** file to configure the maximum number of objects that will be in the **ovwdb** cache. For information on how to do this, see “Configuring **ovwdb**” in chapter 3.
- Change the kernel parameters and regenerate the kernel. For more information, see the “Performance and Traffic” section later in this chapter.

If you exhaust all available memory, the **ovwdb** process will stop running. To verify the status of **ovwdb**, first run the **ovstatus** command to determine if everything is working correctly. If **ovwdb** stops running, **netmon** and **ovtopmd** will also stop running because these processes are dependent on **ovwdb**. If **ovwdb** is no longer running, check the **nettl** log using **netfmt**. Note that this assumes **nettl** was running when the problem occurred. To verify that **nettl** is running, execute the **nettl -status** command. If **ovwdb** stopped running due to memory problems, a disaster message will be logged through **nettl**. To solve this problem, follow the suggestions described above for solving problems with excessive swapping, that is, either edit the **ovwdb.1rf** file or change the kernel parameters.

If the OVW object database gets corrupted in any way, for example, by removing the files in the `/usr/ov/databases/openview/ovwdb` directory, you need to restart the automatic map generation from the beginning. See “Restarting Automatic Map Generation” in chapter 2.

The trapd Background Process

The **trapd** process multiplexes and logs SNMP traps. Basically, **trapd** is an SNMP trap forwarder. When **trapd** receives a trap, it forwards it to all the processes that are registered to receive the trap. Traps are generated by these processes:

- through **ipmap**, when you edit the map or when nodes' status propagates to segments and networks
- **netmon**, when it detects a change as a result of polling nodes or as a result of receiving a trap forwarded by **trapd**
- agent processes, for example, when the HP OpenView SNMP Agent sends SNMP traps from **snmpd**
- **ovtopmd**, when topology changes occur
- **snmpCollect**, when a threshold is exceeded or reset
- configuration operations such as **xnmloadmib**, **xnmtrap**, **xmmsnmpconf**, and **xnmcollect** when you edit configuration files

Figure 4-2 illustrates the interactions between processes and **trapd** when you add an object to a map in HP OpenView Windows. To notify the HP OpenView network management system about the new object, **ovw** sends an **ovw** event to **ipmap**. **ipmap** in turn sends a new node trap to **trapd** and tells **ovtopmd** to update the topology database. **ovtopmd** responds by updating the topology database. **trapd** logs the trap in the **trapd.log** file and forwards the new node trap to **netmon** and **xmmevents**. When **netmon** receives the trap, **netmon** pulls the information it needs from the topology database through **ovtopmd**.

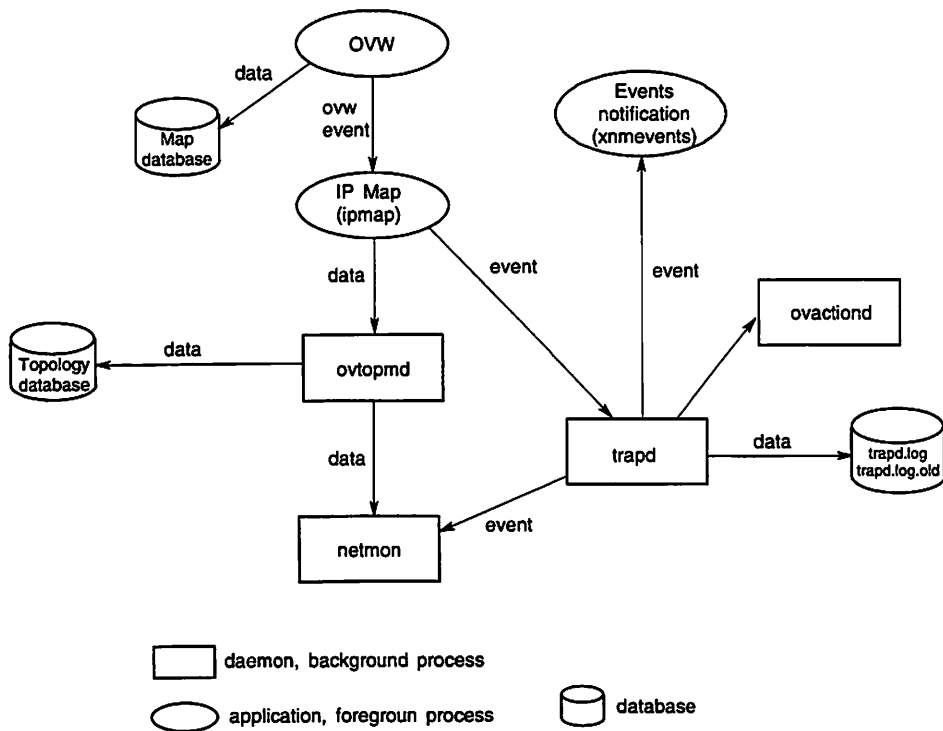


Figure 4-2. trapd Interaction with ipmap Events

SNMP traps generated by an agent (**snmpd** in the case of the HP OpenView SNMP Agent) go to **trapd**, which logs and forwards the traps to **netmon**, **ovactiond**, **ovtopmd**, **ipmap**, and **xnmevents**. Figure 4-3 illustrates this interaction.

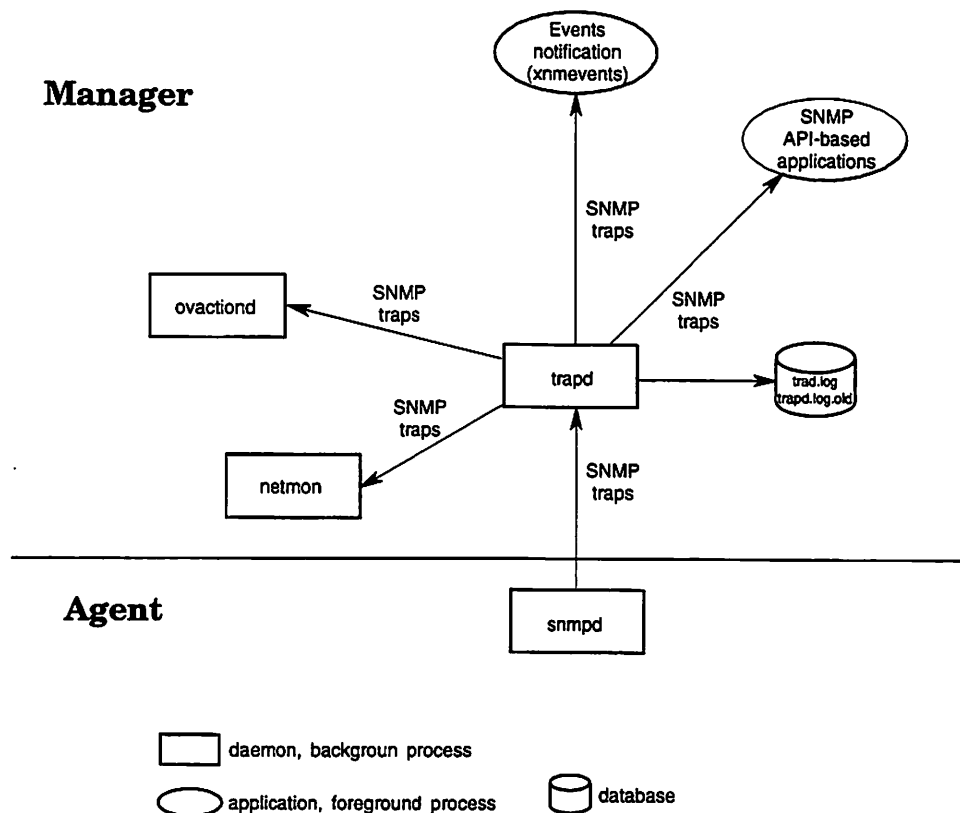


Figure 4-3. trapd Interactions with Agent SNMP Traps

For a discussion of **trapd**'s interactions and operation concerning traps generated by **netmon**, see the section on "The netmon Background Process."

If you suspect a problem with the **trapd** background process, try the following actions to further isolate the problem:

- Check the `/usr/OV/log/trapd.log` file for **trapd** process errors.
- Add a node to the map and then check to see if the event was recorded in the `/usr/OV/log/trapd.log` or `/usr/OV/log/trapd.log.old` files and/or in an **Events Browser** window.
- If the **trapd** process appears to be hung, kill the process and restart all background processes with **ovstart**.
- Turn on tracing as described in the "Recommended Practices" section of this chapter. Your support contact can help you interpret the trace results.

Note

If you are running the HP OpenView network management system on a cluster node, **trapd** accesses the **trapd** running on the cluster server.

If you run **trapd** on a cluster node before running **ovstart** (and hence **trapd**) on the server, **ovw** on the cluster node will fail. It will appear to work, but **ipmap** will not be able to talk to **netmon** so you will not be able to manage or unmanage new objects.

Troubleshooting Operations

This section suggests actions to take if you suspect a problem with one of the operations of the HP OpenView network management system. This section also discusses the sequence of interactions and/or the flow of data associated with an operation when such information may help you to isolate the problem. This section covers the following:

- runtime components
- X Windows components
- network management operations
- browsing MIBs
- building MIB applications and executing MIB applications built by the **MIB Application Builder** operation
- collecting MIB information from network nodes at regular intervals
- configuring events
- loading MIBs
- online help

For detailed information on a command or process mentioned in this section, see its associated man page in the “Reference” appendix.

Runtime Components

If you are having problems running the HP OpenView network management system, you should check file permissions, start-up scripts, and error log. These topics are discussed below.

Manager File Permissions

By default, the **ovw** (graphical network map) program is executable by anyone; the **ovwdb**, **trapd**, **snmpCollect**, **ovactiond**, **ovtopmd**, and **netmon** background processes are executable only by root.

You may restrict the permissions on the HP OpenView Windows (OVW) map files using the **/usr/OV/bin/ovwchmod**, **/usr/OV/bin/ovwchgrp**, and **/usr/OV/bin/ovwchown** commands.¹ These commands change all of the OVW map data files associated with a map at once for you. You use the commands the same way as the operating system **chmod**, **chgrp**, and **chown** commands, except that you follow them with a map name instead of a file name. Note that these commands do not change permissions on the product's directories or executable files, nor on other configuration and data files.

Note Do not change map permissions when OVW is running.

If you are having problems running the HP OpenView Network Node Manager, try restoring the data file permissions of HP OpenView Network Node Manager data file permissions to their default using the **ovwchmod** command with mode 0777.

⁽¹⁾These commands, along with the "**ovwls**" command, are listed on the "**ovwperms (1)**" man page.

Start-Up Scripts

Check to see if some component in the execution sequence is “broken,” such as a syntax error in one of the product’s start-up scripts. Figure 4-4 shows the start-up execution sequence for the HP OpenView network management system. All files are in the `/usr/OV/bin` directory.

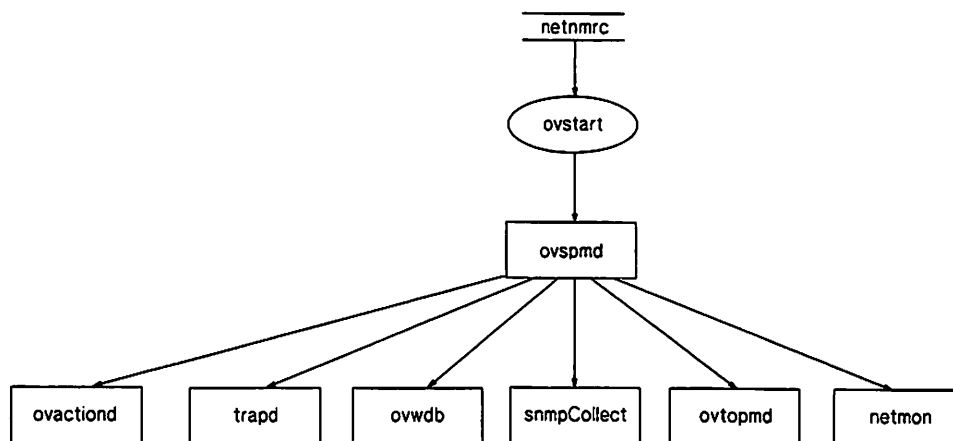


Figure 4-4. Manager Startup

The `/etc/netmrc` file should automatically execute `ovstart` when you reboot the system. If `ovstart` does not execute, the problem may be with the start-up script. The script file is created during installation. If the file already exists, the existing file is moved to `/etc/#netmrc` during installation and the new one is copied to the original file name (`/etc/netmrc`).

Verify that `/etc/netmrc` contains the following:

```
case $NODENAME in
  $ROOTSERVER)
    $SNMPD && echo "$SNMPD \c"
    if [ -x /usr/OV/bin/ovstart ]; then
      /usr/OV/bin/ovstart && echo "ovstart \c"
    fi
  ;;
```

X Window Components

This section discusses X Windows-related files for the HP OpenView Network Node Manager only. If the problem does not appear to be with this product, refer to your X Windows documentation for further troubleshooting information.

To verify the status of the background processes, run the **ovstatus** command. For more information about **ovstatus** and how to troubleshoot the background processes, see “Troubleshooting Background Processes,” earlier in this chapter.

To determine whether there is a problem with the X Window component of the HP OpenView Network Node Manager, try restoring the default X resources for this product, and see if this solves the problem.

1. Ensure that the files **/usr/OV/app-defaults/XNm**, **/usr/OV/app-defaults/OVw**, and **/usr/OV/app-defaults/OVhelp** contain the installed X resource defaults for the product. The OVW and OVHelp X resource defaults are listed in the **ovw** and **ovhelp** man pages. The **XNm** resource defaults are listed in the **XNm** resource file itself.
2. Ensure that the **XENVIRONMENT** shell variable is not set.
3. Remove resources starting with **XNm**, **xmmevents**, **xnmgraph**, **rsam**, **ovw**, or **ovhelp** from your **\$HOME/.xdefaults** file. Or, temporarily rename this file.
4. Restart X Windows.
5. Restart **ovw** (**/usr/OV/bin/ovw**) without any X resources specified as arguments.

You may also want to see this chapter's section on “Network Management Operations” for more information on X Windows-based HP OpenView Network Node Manager operations.

Network Management Operations

If you are having problems running an operation to a remote node, check the following items.

- If you configured agent community names, verify that the node's community name listed in the **Options: SNMP Configuration** dialog box matches the community name configured on that node.
- If you modified any X resources for default network management operation options, ensure that the modifications are correct. You may want to restore the default X resources as explained in this chapter's "X Window Components" section.
- The **Locate Route** operation could fail if one of the nodes in the route uses a routing algorithm that is not based on the BSD (Berkeley Software Distribution) 4.2 routing algorithm or does not support SNMP.

The **Locate Route** operation could also fail if all IP addresses for a gateway do not resolve to the same IP host name.

- Ensure that the remote node supports the operation. Most of the HP OpenView operations require SNMP-based, MIB-I (RFC 1156) or MIB-II (RFC 1158) compliant agent software on the remote node.

The only operations that works to a TCP/IP node without such agent software is:

Diagnose: Network Connectivity -> IP/TCP/SNMP

Diagnose: Network Connectivity -> Ping

Administer: System Management -> HP-UX SAM

Requires HP-UX version 7.0 or later on remote node.

Misc: Terminal Connect -> Telnet (xterm)

Requires ARPA Services on remote node.

Misc: Terminal Connect -> Telnet (hpterm)

Requires ARPA Services on remote node.

Misc: Terminal Connect -> Vt3k (hp3000)

Requires ARPA Services on remote node.

The following operations work only to HP 9000 and Sun systems running the HP OpenView SNMP Agent software:

Monitor: Network Activity -> Ethernet -> Traffic*

Monitor: Network Activity -> Ethernet -> Errors*

Monitor: System Activity -> Disk Space

Diagnose: Network Connectivity -> Remote Ping

Monitor: System Activity -> CPU Load

Monitor: Device Configuration -> SNMP -> Event Notification

Monitor: Network Activity -> SNMP -> Authentication Failures

* These menu items are also active for HP Routers, HP Hubs, HP Bridges, HP 3000/XL, Cisco Routers, and Novell LANtern nodes.

Execution Models for Network Management Operations

Some of the HP OpenView's operations start directly from **ovw**; others are started by an integrating process, **/usr/OV/bin/xnmappmon**.

Figure 4-5 shows those network management operations started (or startable) directly from **ovw**.

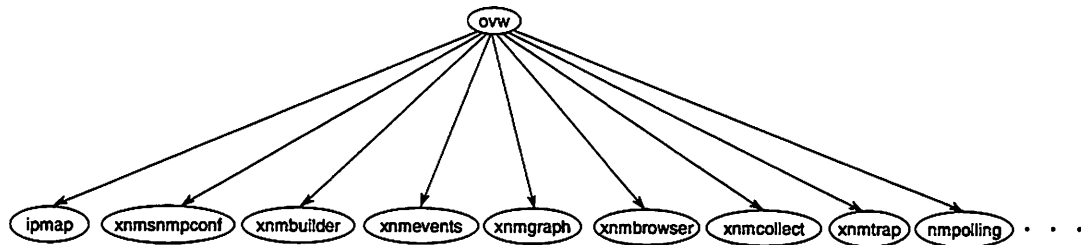


Figure 4-5. Applications Started Directly from OVW

Figure 4-6 shows those operations started by **xnmappmon**.

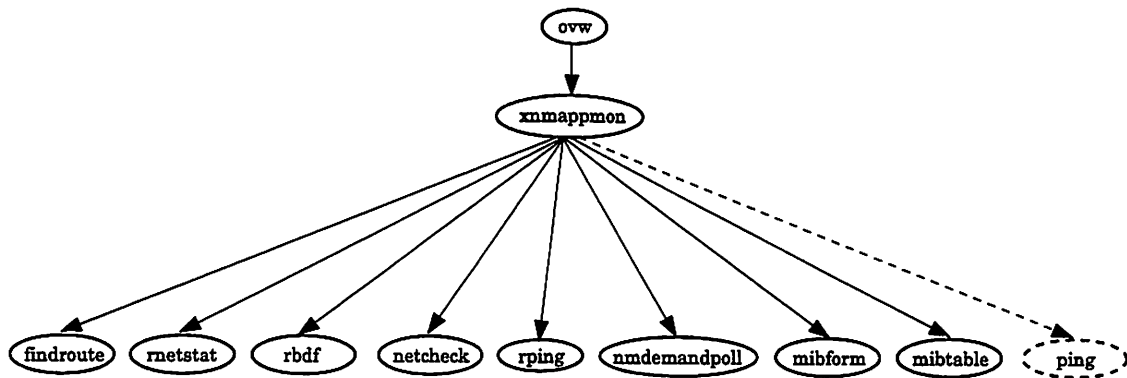


Figure 4-6. Applications Started by xnmappmon

Many of the processes shown in figures 4-5 and 4-6 may be run as commands for remote troubleshooting. For information about which commands are executed for the different processes, see the corresponding man page. **ping** is shown as a dotted line because it is a system command.

Browsing Internet MIBs

If you have problems browsing the Internet MIB using the **Monitor: MIB Values -> Browse MIB** operation, check the following:

- Make sure the Internet MIB is loaded. Use the **Options: Load/Unload MIBs** operation to see what modules are currently loaded. If the MIB is not listed, use the **Load/Unload MIBs** operation to load the Internet MIB into the MIB description file.
- Make sure that the version of the Internet MIB loaded into the HP OpenView network management system matches the Internet MIB version that is implemented on the device being managed. When you get a new device on your network or upgrade an existing one, make sure that you get the Internet MIB for that device.

If you are unable to set the value on an object so you can physically update the device, make sure that the agent is configured to respond to SNMP SetRequests. To do so, enter a community name that will allow SetRequests on the node in the **Options: SNMP Configuration** operation or enter the community name within the **Browse MIB** dialog box. For more information about the community name, see “Agent Community Names” in chapter 3.

Building and Executing MIB Applications

The problems you may encounter while building MIB applications and executing MIB applications built by the **Options: Application Builder -> SNMP** are closely related.

For example, some errors made while building an application may not show up until you actually execute the application. If the problem is an incorrectly built application, you need to modify your application.

This section is organized as follows:

- Things to check when you have problems building an application.
- Things to check when you have problems executing an application. Problems executing applications can be caused by either an error in the way the application was built or by independent things such as the community name not being configured correctly.

Building MIB Applications

If you have problems building MIB applications using the **MIB Application Builder** operation, check the following:

- Make sure to read the instructions in online help about adding MIB applications.
- Make sure you have the enterprise-specific MIB loaded. Use the **Load/Unload MIBs** operation to see what modules are currently defined. If the MIB is not listed, use the **Load/Unload MIBs** operation to include the MIB in the Loaded MIB database.
- Make sure the MIB object ID you select is supported by the device you are trying to monitor. To verify this, use the **Browse MIB** operation.
- When you create a “Table” application, you can only select fields from a single table in the MIB tree.
- Make sure that the version of the MIB loaded into the HP OpenView network management system matches the MIB version that is implemented on the device being managed. When you get a new device on your network or upgrade an existing one, make sure that you get the MIB for that device.

Executing MIB Applications

If you have problems executing applications built by the **MIB Application Builder** operation, check the following:

- Make sure you have the enterprise-specific MIB loaded. The application does not come up when the MIB for that particular application is not loaded. Use the **Load/Unload MIBs** operation to see what modules are currently defined. If the MIB is not listed, use the **Load/Unload MIBs** operation to load the MIB into the MIB description file.

The MIB may be missing, for example, if you load a MIB, build an application, and then unload the MIB or if you copy the application to a new machine and the MIB is not on that machine.

Make sure that if you copy the MIB application to another machine, you also load the enterprise-specific MIB on that machine.

- Make sure the MIB object ID you select is supported by the device you are trying to monitor. To verify this, use the **Browse MIB** operation.
- Make sure the application was built to monitor the device you have selected. The problem may be that the application was built to monitor a different device than the one you are trying to monitor. For example, the application was built for an HP-UX agent and you are trying to run that application on a Fred Router from Flintstones Company.
- Verify that the management station can talk to the agent. Use the **Browse MIB** operation to verify that the manager station can actually talk to the agent and that the agent is responding with the values. Also, use the **Diagnose: Network Connectivity -> IP / TCP / SNMP** operation.
- Verify that the community name is set up properly in the **Options: SNMP Configuration** operation. If you get a message indicating no response from the device, the manager station may not have the proper community name defined for that device.
- Make sure that the version of the MIB loaded into the HP OpenView network management system matches the MIB version that is implemented on the device being managed. When you get a new device on your network or upgrade an existing one, make sure that you get the MIB for that device.
- Verify that the following three files created by the **MIB Application Builder** are intact:
 - The registration file in `/usr/OV/registration/$LANG/ovmib`, where `$LANG` is the language the HP OpenView network management system is set to use. Make sure the registration file name for the application is the same as the Application Name that was specified when the application was created.
 - The help file in `/usr/OV/help/$LANG/ovmib/OVW/functions`. The help file name is the same as the Application Name that was specified when the application was created.
 - The line entry in the OVW help index file in the `/usr/OV/help/$LANG/OVW/functions/INDEX` directory corresponding to your specific MIB application.
- If you typed the object identifier in the MIB selection field, make sure you enter a full object identifier. For example, you cannot create a form application for `.iso.org.doc.internet`. The object ID must correspond to an object supported by the selected device.

Troubleshooting Data Collector Problems

Figure 4-7 shows the interactions among the processes `xnmCollect`, `xnmgraph`, `snmpCollect`, and `snmpColdDump`; and the agents.

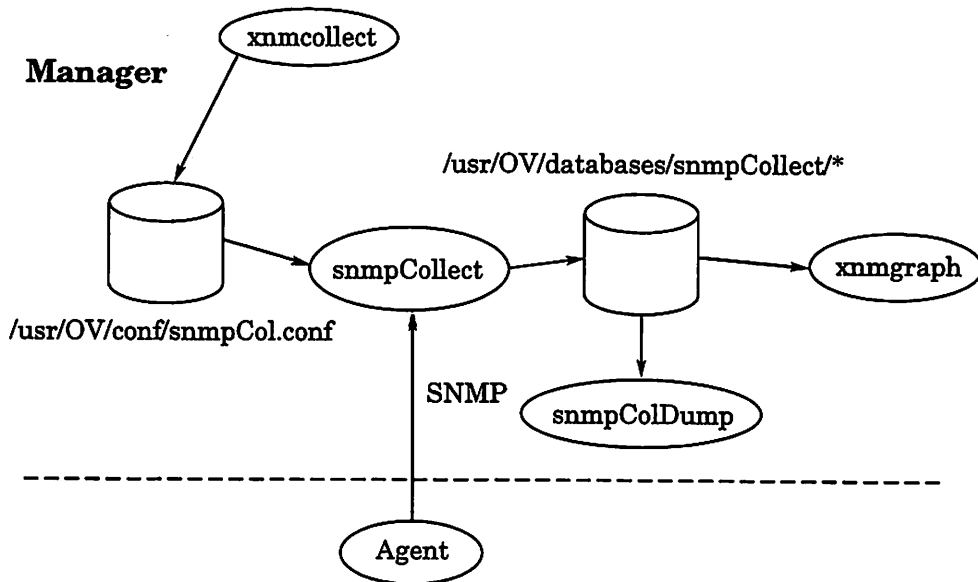


Figure 4-7. snmpCollect Interaction with SNMP Agents

If you have problems collecting MIB data, check for the following problems.

- The process for the data collector, **snmpCollect**, may not be running on the manager station.
 - Make sure that you are not out of disk space. The data collector stops running when the disk is full. See the next page for information about what to do if the disk is full.
 - Check that the **snmpCollect** process is running by executing the **ovstatus snmpCollect** command. If the **snmpCollect** process is not running, as root restart the data collector using the **ovstart** command.
- Data is not being collected from the node it should be or thresholds are not being generated.
 - Make sure the target node supports SNMP.
 - Verify that the MIB object is configured for collection. Use the **MIB Data Collection** operation to verify the configuration.
 - Verify that the target node is configured to collect data or generate threshold events. You can configure the node to collect data either by explicitly specifying the node name or by specifying a wildcard. Also, verify that the target node has not been excluded for collection. Use the **MIB Data Collection** operation to verify the configuration.
 - See if other collections or thresholds from that node are working. If none are working, the problem is probably with the node. From the **MIB Data Collection** dialog box, select the MIB object and then click on the **Show Data** button to see if data is being collected from the node.

- Try to access the MIB object using the **Browse MIB** operation.
 - Run **snmpCollect -s** and look at the **/usr/OV/log/snmpCol.trace** file. This command lists the configuration of the data collection.
 - Run **snmpCollect -t** to toggle the tracing. To determine whether or not tracing is on, look at the trace file. See the **snmpCollect (1M)** man page for more information.
 - Try turning on verbose tracing using **snmpCollect -v**.
 - For more information about troubleshooting threshold events, see the section “Configuring Events.”
 - If you expect threshold events to occur, but they do not, start a data collection on the MIB variable. The problem may be that the threshold value is not set right. For example, if the normal value for this variable is 20 and you set the threshold value to 100, you will probably never see a threshold event.
- The disk is full. The data collector stops running when the disk is full. To solve this problem and to prevent this problem from happening again, do the following:

To solve the problem:

- Free up disk space by removing files in the **/usr/OV/databases/snmpCollect** directory. Note, however, when you remove files you lose the data you have saved. If you don't want to remove the whole file, you can edit the file and keep only the most recent data by deleting older entries in the file. Once you have freed up disk space, as root restart the data collector with the **ovstart** command.
- Edit files in the **/usr/OV/databases/snmpCollect** directory using the **snmpColDump** command. For example, to retain the last 100 entries of a file, use the following commands:

```
snmpColDump -tTI /usr/OV/databases/snmpCollect/FILE | \
    awk -F\t '{printf("%d\t%d\t%s\t%lg\n", $4, $5, $6, $3)}' | \
    tail -100 > /tmp/save
```

```
snmpColDump -r /tmp/save /usr/OV/databases/snmpCollect/
```

- If you want to stop all data collection, kill the **snmpCollect** process with the **ovstop snmpCollect** command or suspend all data collections using the **MIB Data Collection** operation.

To prevent the problem in the future:

- Reduce the polling intervals. Using the **MIB Data Collection** operation, set the polling interval to not collect as often. In addition, if you only want to check thresholds, don't store the data.
- Reduce the number of MIB objects you are collecting.
- Reduce the number of nodes you are collecting on.
- Set up **cron (1M)** to periodically remove files in the **/usr/OV/databases/snmpCollect** directory.

If you can't resolve the problem, use the tracing command. To use the tracing command, try restarting the data collector with the same command as in **/usr/OV/lrf/snmpCol.lrf** file, but add the **-t** option. The **-t** option is a tracing option that logs the results to the **/usr/OV/log/snmpCol.trace**. Run

snmpCollect -T to toggle the tracing. To determine whether or not tracing is on, look at the trace file. See the **snmpCollect (1M)** man page for more information.

Configuring Events

If you have problems configuring events using the **Options: Event Configuration: SNMP** operation, first see the vendor documentation that describes the traps generated by that specific device to make sure the enterprise and trap number correspond to what you have indicated.

To get a description of the HP OpenView events shipped with the product, use the **Options: Event Configuration: SNMP** operation. From the **Event Configuration** dialog box, select **OpenView** as the **Enterprise Name**, then select an **Event Name** and click on **Describe**.

If you have defined an action for an event, but the action is never taken or it is not executed the way it was intended to, check the following:

- Verify the configuration. When you save the configuration for a new event, the entire **trapd.conf** file is rewritten and an event is sent to **xmmevents**, **ovactiond**, and **trapd**. Common configuration problems are
 - the command is typed wrong
 - the arguments are not specified properly
- Make sure the format message for a particular trap is intact. For example, if the format message for the coldStart trap has been erased, you will not see that trap in the event log file or the **Events Browser** window.
- Make sure you do not edit the **trapd.conf** file. The **trapd.conf** file contains the logging and action definitions. If this file is corrupted, the events you have configured may not be logged.
- Compare your **/usr/OV/conf/C/trapd.conf** file with the default configuration file, **/usr/OV/newconfig/OVSNMP-RUN/c/trapd.conf**.
- Check the **trapd.log** file to see if the trap was decoded properly. Each time the HP OpenView network management system receives an event, the **trapd** process logs an entry in the **trapd.log** file. For example, the **trapd.log** file identifies ASN.1 coding problems in enterprise-specific traps.
- Verify that the event is logged to the expected category. For example, if you selected log only, you will not see the event in the **Events Browser** window. If you selected ignore, you will not see the event.

Loading Internet MIBs

When you load an Internet MIB, the **Options: Load/Unload MIBs** operation takes all vendors' Internet MIBs and puts them into one single Loaded MIB database, `/usr/ov/conf/snmpmib`. The **Load/Unload MIBs** operation creates a binary file, `/usr/ov/conf/snmpmib.bin`. The **Browse MIB**, **MIB Data Collection**, **MIB Application Builder**, and the MIB applications built by the **MIB Application Builder** access the Loaded MIB database for their routines

Note

Always use the **Load/Unload MIBs** operation to modify the Loaded MIB database; do not edit the Loaded MIB database directly.

If you have problems including Internet MIBs in the MIB description file using the **Load/Unload MIBs** operation, check the following:

- Make sure that the format of the MIB definition is correct. The format must conform to RFC 1155 or RFC 1212. For example, it must have an ASN.1 module DEFINITIONS clause. The enterprise-specific files should only contain modules that are unique to that vendor.
- Make sure that the syntax is correct. Hewlett-Packard has verified that the syntax of the MIB modules included with the HP OpenView product is correct. However, if you get the MIB module directly from the vendor, you may have to correct syntax errors. The **Load/Unload MIBs** operation checks for syntax errors and displays an error that tells you where the error is and how to fix it. The most common syntax errors are typing errors and non-conformance to the RFCs.
- Compare the MIB with the Hewlett-Packard MIB to look for errors. By comparing the MIB that does not load with the Hewlett-Packard MIB, you may be able to troubleshoot the problem. The Hewlett-Packard MIB loads correctly.

Known Limitations

The object label in a MIB module must be unique. Due to lack of standards this is not clearly identified by SNMP. If you have mnemonic names for an object ID component that is used elsewhere in the module, the second object shows up in the MIB tree under the first object that has the common component name. Furthermore, the translation from the mnemonic object name to the numeric object ID will be incorrect, causing SNMP queries to fail.

EXAMPLE:

Suppose the xyzCorp has the enterprise MIB **device** object defined in a module XYZCORP-MIB under **.iso.org.dod.internet.private.** as

```
enterprises.xyzCorp.bridge.device.name
                                .number
                                .ROMId
```

and another **device** object defined as

```
enterprises.xyzCorp.hub.device.model
                                .serialNumber
                                .otherInfo
```

The MIB parser cannot distinguish between the two **device** objects. As a result, the xyzCorp fields **model**, **serialNumber**, and **otherInfo** will appear incorrectly in the Internet MIB tree as

```
enterprises.xyzCorp.bridge.device.model
                                .serialNumber
                                .otherInfo
```

To circumvent the problem, patch the enterprise-specific MIB to use a unique component name. For the example above, change **~.hub.device** to **~.hub.hubdevice**. When the object name is translated to the numeric object ID used in the SNMP query, the proper object ID will be used and the SNMP query should work.

A second (and more contrived) alternative is to put the second and any subsequent occurrences of duplicate object labels (and any objects based on them) in a separate ASN.1 module with a fully qualified reference as to their proper position in the global Internet MIB tree.

For example, the first occurrence of **device** could be defined in a module as

```
XYZCORP1-MIB DEFINITIONS ::= BEGIN
xyzCorp OBJECT IDENTIFIER ::= { iso org(3) dod(6) internet(1) private(4)
                                enterprises(1) 999 }
bridge OBJECT IDENTIFIER ::= { xyzCorp 1 }
device OBJECT IDENTIFIER ::= { bridge 1 }
-- all subsequent children...
.
.
END
```

and the second in a module as

```
XYZCORP2-MIB DEFINITIONS ::= BEGIN
hub OBJECT IDENTIFIER ::= { iso org(3) dod(6) internet(1) private(4)
                             enterprises(1) xyzCorp(999) }
device OBJECT IDENTIFIER ::= { hub 1 }
-- all subsequent children...
.
.
END
```

Online Help

The text files used for the HP OpenView Windows' **Help** operations (not including man pages) are below the directory **/usr/OV/help/\$LANG**.

If the map (**ovw**) is not running, you can view these files using operating system commands such as **more**. Also, this manual's "Reference" appendix contains copies of the man pages, or you can use the **man** command on the man page name.

Performance and Traffic

The performance of the HP OpenView network management system is limited primarily by three factors:

- The amount of memory installed in the manager system.
- The manager system's processor (CPU) speed.
- The performance of X Windows.
- For detailed information on performance of the HP OpenView network management system, refer to the *HP OpenView SNMP Platform Performance and Configuration Guide, with HP OpenView Network Node Manager Examples*.

Performance

To increase performance, you may want to install additional memory and/or upgrade your CPU.

You can also, in some cases, increase performance by minimizing file system swapping. OVW performance is best when there is no file system swapping required. With no file swapping, **ovwdb** is running as efficiently as possible; all the object information it is managing is residing in physical memory. Any system swapping that must be done is affected by two factors: the number of objects maintained in physical memory and the process size limit.

- If you have a system where OVW performance is poor due to file system swapping, you may use the “-n” option for **ovwdb** to increase the maximum number of objects that **ovwdb** can keep in physical memory, while storing the rest in a database. This option lets **ovwdb** store more objects than would normally fit in the process database space.

Note that this option is not always an appropriate solution for memory problems due to file swapping, and it is not a good performance enhancement, because **ovwdb** database queries are often random. For example, a **Locate** operation will cause a query over many objects in searching for a field value. In this operation, using the “-n” option would cause **ovwdb** to update the cache with each object involved in the search as well as with all the field values for those objects. The best way to correct poor performance due to file system swapping is to increase the amount of physical memory available to **ovwdb**. More physical memory reduces the amount of swapping needed.

If the “-n” option is not used, **ovwdb** will continue to grow as new objects are created until either

- the maximum process size limit is reached.
- the system swap space is exhausted.

The **ovwdb.1rf** file is shipped with a default limit of 5000 objects set for this option.

- Another factor that can limit the number of objects managed by **ovwdb** is the process size limit. If the maximum process size limit is reached, **ovwdb** will terminate. For information on increasing the process size limit, see the appropriate system administration guide.

Traffic

Status polling generates the most network traffic. You can monitor the traffic generated by the HP OpenView network management system by using the **netstat -i** command. Run the **netstat -i** command several times and observe the differences in the number of packets sent and received. Be sure that the manager system is not generating any other outbound traffic. Note that the command used to show the number of packets sent and received may be different depending on the operating system you use. Make sure that you are using the appropriate command.

You can regulate the traffic generated by the HP OpenView network management system through the following methods.

- Adjust polling intervals through the **Options: SNMP Configuration** or **Options: Topology/Status Polling -> IP** operation. Note that reducing the amount of polling traffic by lengthening polling intervals may delay real-time map updates, resulting in a less accurate map.
- Unmanage nodes with the **Options: Unmanage Objects** operation such that you are monitoring only what really needs to be monitored. Note that this tactic decreases the amount of available management information for unmanaged nodes.
- Adjust the amount of MIB data that is collected through the **Options: Data Collection & Thresholds: SNMP** operation. You can
 - Stop collecting MIB data on some of the MIB objects that are configured to collect data by changing the status to **Suspend**.
 - Modify the details of the configured MIB objects by either deleting nodes, excluding nodes, or changing polling intervals.

If the HP OpenView network management system starts to use excessive swap space, performance may decrease. To check the size of the HP OpenView Windows program, **ovw**, (and other processes), use the **ps -l** command. If the process grows too large, exit HP OpenView network management system and restart it. (The program automatically saves the map between invocations.)

netmon will generate some network broadcast traffic; how much traffic depends on several factors.

- If you use this command without the “-J” option, **netmon** generates, if necessary, just an ARP Request. There are several factors that determine whether **netmon** needs to generate an ARP Request.
 - For nodes that are up, but not on the same subnet where the management station is, **netmon** generates just one ICMP echo request packet and gets back one response. In this case, no ARP Request is generated.

Because the management station is on a different subnet, a request will go through a gateway.

Gateways always have large ARP caches. **netmon** knows the gateway's link-level address and does not have to broadcast an ARP Request to that gateway. **netmon** only has to send one ICMP echo request through the gateway and gets back one response.

- For devices that are on the same subnet as the management station, **netmon** may generate just one ICMP echo request, or it may also generate one ARP Request as well. Because the nodes are on the same subnet, **netmon** does not have to go through a gateway. Broadcasting an ARP Request will depend on whether or not **netmon** has in its own ARP cache the device's link-level address.
- **netmon** will generate an ARP Request based on whether or not the IP address **netmon** is trying to do a status polling of is still in the ARP cache of that system.

The presence of the IP address in the ARP cache is affected by how many nodes are on the subnet and by how much physical memory is in the system (some systems will automatically adjust the size of their ARP table).

- If invoked with the “-J” option, **netmon** may cause up to three broadcast ICMP echo requests to be broadcast on each subnet it discovers. These broadcasts will only be issued on a subnet if
 - No broadcast ICMP echo request has ever been issued on this subnet on behalf of this manager station.
 - An agent capable of issuing a broadcast ICMP echo request on **netmon**'s behalf has been discovered on this subnet. All HP-UX SNMP agents have this capability.
- The amount of **netmon**'s broadcast traffic also depends on your network's configuration:
 - How many nodes are up and how many are down.
 - How many nodes are on the same subnet.
 - How many nodes are on other subnets.
 - How large the permanent ARP cache is.
 - How probable can a particular IP address be found in the permanent ARP cache.

Upgrading from NNM 2.x to NNM 3.x

This appendix lists the major differences between NNM 2.0 and 2.1 releases and the NNM 3.x releases. It does not discuss new functionality. The appendix includes the following:

- backing up NNM 2.x
- file placement and formats
- menu structure

Backing up NNM 2.x

Before you update from NNM 2.x to NNM 3.x, make sure that you back up your existing NNM 2.x environment. Specifically, back up or make a tar image file of the following:

- the `/usr/etc/nm` directory
- the `/usr/adm/snmpCollect` directory
- the `/usr/adm/trapd.log` file (optional)

If you need to restore your NNM 2.x environment, follow these steps:

1. Execute the `ovstop` command to stop all NNM 3.x background processes.
2. Remove the NNM 3.x installation using the `ovremove` command as follows:

```
ovremove -p NNMGR
ovremove -p SNMFRUN
```

3. Reinstall NNM 2.x from the NNM 2.x product tape.
4. Restore the NNM 2.x data files that you backed up.

File Placements and Formats

The file placements have changed from the HP OpenView Network Node Manager (NNM) 2.x releases. In the 3.x release all files are placed under the `/usr/ov` directory.

Most of the files are automatically converted and upgraded from the 2.x configuration to NNM 3.x through conversion utilities. However, not all files are converted. This section

- explains how to convert the files that require manual conversion
- lists the files that are automatically converted

Files Requiring Manual Conversion

You need to manually convert the following files

- `netmon` threshold configuration
- “Drop In” applications
- `oid_to_type` file

netmon Threshold Configuration

Threshold monitoring by `netmon` for the following seven thresholds is obsoleted in NNM 3.x

- Interface Percent Deferred
- Interface Percent Collisions
- Interface Percent Input Errors
- Interface Percent Output Errors
- Interface CRC Errors
- CPU Load
- Disk Space Percentage Used

You need to manually redefine these thresholds using the

Options: Data Collection & Thresholds -> SNMP operation as follows:

1. From the OVW menu, select **Options: Data Collection & Thresholds -> SNMP...**; the **MIB Data Collection** dialog box appears.
2. Click on **[Add...]**; the **MIB Data Collection/MIB Object Selection** dialog box appears.
3. Select the threshold that you want to monitor.

To define thresholds for CPU Load and Interface CRC Errors, move down the MIB tree to select the appropriate MIB object. See the following table.

To redefine NNM 2.x threshold...	Select the following MIB object...	Instance
CPU Load	<code>iso.org.dod.internet.private.enterprises.hp.nm.system.general.computerSystem.computerSystemAvgJobs1</code>	0
Interface CRC Errors	<code>iso.org.dod.internet.private.enterprises.hp.nm.interface.ieee8023MacTable.ieee8023MacEntry.ieee8023MacCRCErrors</code>	.*

To define thresholds for Interface Percent Deferred, Interface Percent Collisions, Interface Percent Input Errors, Interface Percent Output Errors, and Disk Space Percentage Used, do the following:

- a. Click on the Expression button; the selection list shows currently available MIB expressions.
- b. Select the appropriate expressions. See the following table.

To redefine NNM 2.x threshold...	Select the following Expression...	Instance
Interface Percent Deferred	<code>If%deferred</code>	.*
Interface Percent Collisions	<code>If%collision</code>	.*
Interface Percent Input Errors	<code>If%inErrors</code>	.*
Interface Percent Output Errors	<code>If%outErrors</code>	.*
Disk Space Percentage Used	<code>Disk%util</code>	.*

4. Set up your target nodes to monitor.
5. Select the threshold mode by choosing one of the following from the option menu next to the Collection Mode label:
 - Don't Store, Check Thresholds
 - Store, Check Thresholds

For more details on how to set up thresholds, see the online help available from the **Options: Data Collection & Thresholds -> SNMP** dialog box.

“Drop In” Applications

NNM 2.x “drop in” application registration files (that is, type III applications) are *not* converted to the HP OpenView Windows 3.x format automatically. However, a conversion utility is provided with HP OpenView Windows 3.x that assists in the conversion process. To convert existing “drop in” applications, follow these steps:

1. Log on as root.
2. Use the **cvtRegFile** conversion utility to convert your NNM 2.x registration file to the HP OpenView Windows 3.x format. The syntax of the **cvtRegFile** conversion utility is

```
cvtRegFile [-list] [<nnm2.Xregfile>...]
```

or

```
cvtRegFile [-stdout] [-appdir <dirname>] <nnm2.Xregfile>...
```

The options are

- | | |
|----------------|--|
| -list | Lists the specified NNM 2.x registration files that can be converted (that is, Type III only). |
| -stdout | Prints the results of registration file conversion to standard output instead of creating a file in the OVW 3.x registration directory. The -stdout option enables you to see what the OVW 3.x registration file would look like without actually creating it. |
| -appdir | <dirname> overrides where to store the converted registration file. If <dirname> is a relative path, it specifies an application subdirectory under the standard OVW 3.x registration directory, /usr/OV/registration/\$LANG . If <dirname> is a full path, (that is, it begins with /), then it is used instead of the OVW registration directory. |

nnm2.Xregfile The name of the NNM 2.x registration file or files to convert.

You may want to first use the **-list** option to find all the NNM 2.x registration files that are Type III. To do so, type

```
cvtRegFile -list
```

Then convert only the ones that you want to convert, omitting the Type III applications that might be part of other HP OpenView products that already have equivalent registration files in the NNM 3.x environment.

The main menu bar items are converted to the HP OpenView Style Guide conventions. For example, menu items under the **Request** menu are automatically converted to the **Monitor** menu, and menu items under **Control** are converted to the **Administer** menu.

3. Optionally, edit the HP OpenView Windows 3.x formatted registration file to modify or add OVW-specific definitions (for example, SelectionRule, Name Field, Description, and Version). Note that if you edit the menu path, avoid creating new items in the OVW main menu bar. Also, avoid creating menu cascades more than three levels deep as per the HP OpenView Style Guide recommendations.

oid_to_type File

The location and format of this file has changed since the 2.x releases. Entries in the NNM 2.x **oid_to_type** file now are divided among two NNM 3.x files: **oid_to_type** and **oid_to_sym**. Any entries you edited in the NNM 2.x **oid_to_type** file must be manually added to or changed in the two NNM 3.x files.

Files Automatically Converted

This section lists the files that are automatically converted and upgraded to NNM 3.x.

- **IP Map databases**

A conversion utility is provided with HP OpenView Windows 3.x that converts NNM 2.x map databases to the HP OpenView Windows database format and location. The new databases are in the **/usr/OV/databases/openview** directory. The NNM 2.x map databases are renamed and left in **/usr/etc/nm/databases/topo_db.old**. Note that if you are converting more than 1,000 nodes, the conversion will not happen automatically. Follow the instructions in **/tmp/update.log**.

NOTE:

NNM 2.x Map Snapshots cannot be converted to the HP OpenView Windows 3.x format.

- **SNMP Community Name Configuration**

The NNM 2.x community name configuration file **/etc/community** is converted to the enhanced SNMP configuration file **/usr/OV/conf/ovsnmp.conf** during installation.

- **Event Formats & Actions**

The location of the event format & actions configuration file has changed. The configuration file format remains the same. If an NNM 2.x version of the file **/usr/etc/nm/conf/trapd.conf** exists, it will be moved into the new location **/usr/OV/conf/C/trapd.conf** during installation.

- **Compiled MIB Definitions**

The binary format and file location of loaded MIB definitions have changed. The MIB definitions loaded in NNM 2.x **/usr/etc/nm/conf/mib** are re-compiled and loaded in the new NNM 3.x format **/usr/OV/conf/snmpmib** during installation.

- **Source MIB Definitions**

The MIB definition sources in the NNM 2.x directory **/usr/etc/nm/mibs** are copied to the NNM 3.x directory **/usr/OV/snmp_mibs**. The **mib-2** file is renamed to **rfc1213-MIB-II**.

- **Data Collection/Threshold Configuration**
NNM 2.x data collection and threshold configuration formats in `/usr/etc/nm/conf/snmpCol.conf` are converted into the new 3.x file format and location `/usr/OV/conf/snmpCol.conf` during installation.
- **Data Collection Databases**
NNM 2.x data collection flat-file databases in the `/usr/adm/snmpCollect` directory are converted into the new 3.x file format and placed in the `/usr/OV/databases/snmpCollect` directory during installation.
- **netmon Polling Intervals**
The file location for the `netmon` polling intervals is automatically changed; the format remains the same.
- **MIB Applications**
NNM 2.x MIB application definitions, built by the **MIB Application Builder**, are converted from `/usr/etc/nm/registration/C` into the HP OpenView Windows 3.x registration format and moved to `/usr/OV/registration/OV/ovmib/*` during installation. Help files are also converted and moved from `/usr/etc/nm/help/C/ihelp` to `/usr/OV/help/C/ovmib/OVW/Functions`.

Furthermore, the MIB applications will be converted to take advantage of the enhanced graphing and common application interfaces in NNM 3.x.

Menu Structure

The following table maps the NNM 2.x menu structure into the NNM 3.x menu structure. Note that NNM 2.x menu items not listed in this table are incorporated in HP OpenView Windows (OVW). For a listing of OVW menu items, see the *HP OpenView Windows User's Guide*.

Note The LANtern and Cisco applications are not automatically installed by default. Therefore, you will not see these menu items unless you install these menu applications.

NNM 2.x Menu Location

NNM 3.x Menu Location

Options Menu

Options: Load/Unload MIBs

Options: Load/Unload MIBs: SNMP...

**Options: MIB Application
Builder**

Options: Application Builder: SNMP...

Options: MIB Data Collector

**Options: Data Collection & Thresholds:
SNMP...**

**Options: Trap Formats &
Actions**

Options: Event Configuration: SNMP...

Request Menu

Monitor Menu

Request: Events

Monitor: Events -> SNMP...

Request: Interfaces

**Monitor: Network Configuration ->
Interfaces...**

Request: Addresses

**Monitor: Network Configuration ->
Addresses...**

Request: Routing Table

**Monitor: Network Configuration ->
Routing Table...**

Request: ARP Cache

**Monitor: Network Configuration ->
ARP Cache...**

NNM 2.x Menu Location

Request: TCP Connections

Request: Disk Space

Request: Services

Monitor Menu

**Monitor: MIB Values-> Browse
MIB**

**Monitor: MIB Values->LANtern->
MAC to IP Addresses**

**Monitor: MIB Values->LANtern->
Duplicate IP Addresses**

Monitor: Interface Traffic

**Monitor: MIB Values -> Cisco->
Interface Performance**

Monitor: Link Receptions

Monitor: Link Transmissions

**Monitor: MIB Values->LANtern->
Error Packets**

**Monitor: MIB Values->Cisco->
Interface Errors**

**Monitor: MIB Values->LANtern->
Packet Types**

**Monitor: MIB Values->LANtern->
Packet Sizes**

**Monitor: MIB Values->Cisco->
Traffic Matrix**

NNM 3.x Menu Location

**Monitor: Network Activity -> TCP:
Connections...**

Monitor: System Activity -> Disk Space

**Monitor: Network Configuration ->
Services...**

Monitor: MIB Values->Browse MIB: SNMP...

**Monitor: Network Configuration ->
Addresses**

**Monitor: Network Configuration ->
Duplicate Addresses**

**Monitor: Network Activity ->
Interface: Traffic...**

**Monitor: Network Activity ->
Interface -> Traffic...**

**Monitor: Network Activity -> Ethernet ->
Traffic...**

**Monitor: Network Activity -> Ethernet ->
Errors...**

**(Functionality combined with Monitor: Net-
work Activity -> Ethernet -> Traffic... and
Network Activity -> Ethernet -> Errors...)**

**Monitor: Network Activity -> Ethernet ->
Errors...**

**Monitor: Network Activity -> Ethernet ->
Errors...**

**Monitor: Network Activity -> Ethernet ->
Packet Types**

**Monitor: Network Activity -> Ethernet ->
Packet Sizes**

**Monitor: Network Activity ->
Ethernet -> Traffic Matrix**

NNM 2.x Menu Location

Monitor: MIB Values->LANtern->
Node Statistics

Monitor: MIB Values->Cisco->
Accounting Summary

Monitor: MIB Values->Cisco->
Configuration->General

Monitor: MIB Values->LANtern->
Configuration->General

Monitor: MIB Values->Cisco->
Configuration->TTY Lines

Monitor: MIB Values->LANtern->
Configuration->Serial Lines

Monitor: MIB Values->LANtern->
Alarm Recipients

Monitor: MIB Values->LANtern->
Client Access

Monitor: MIB Values->Cisco->
Buffer Utilization

Monitor: MIB Values->LANtern->
Alarm Statistics

Monitor: CPU

Control Menu

Control: HP-UX SAM

Test Menu

Test: Ping

Test: Remote Ping

Test: Protocol->All

Locate Menu

Locate: Route

NNM 3.x Menu Location

Monitor: Network Activity ->
Ethernet -> Traffic Matrix

Monitor: Network Activity -> IP -> Errors

Monitor: Device Configuration -> General

Monitor: Device Configuration -> General

Monitor: Device Configuration ->
Serial Lines

Monitor: Device Configuration ->
Serial Lines

Monitor: Device Configuration -> SNMP ->
Access Authorization -> SNMP

Monitor: Device Configuration -> SNMP ->
Event Notification...

Monitor: Device Activity -> Memory Usage

Monitor: Device Activity -> Thresholds
Exceeded

Monitor: System Activity -> CPU Load...

Administer: System Management: HP-UX SAM...

Diagnose: Network Connectivity -> Ping

Diagnose: Network Connectivity ->
Remote Ping...

Diagnose: Network Connectivity ->
IP / TCP / SNMP...

Diagnose: Network Connectivity ->
Locate Route: Via SNMP

Creating and Using Registration Files

This appendix describes the Registration Files used to configure applications within HP OpenView Windows (OVW). Registration files communicate a wide variety of information to OVW, such as how applications are integrated into OVW and how graphical symbols are defined. For information about the Local Registration Files (LRF), see chapter 3 “Configuration.”

Appendix Overview

This appendix explains:

- How to integrate applications into the HP OpenView Windows user interface using Application Registration Files.
- How to define graphical symbols using Symbol Type Registration Files.
- How to define fields for objects using Field Registration Files.

This appendix introduces the Application Registration Files, complete with some simple examples. If you need more information, see the **OVWRegIntro(5)** man page in the “Reference” appendix. A complete syntactical description of all registration files is provided in the **OVWRegIntro(5)** man page.

The registration files provide a static mechanism to configure OVW behavior.

Note

If you are developing applications and want to integrate your applications more tightly into OVW, you can use the HP OpenView Windows API. The **ovw** API is available with the HP OpenView SNMP Developer's Kit.

Application Registration Files

Application registration files are used to integrate network and systems management applications with the HP OpenView Windows (OVW) user interface. Many aspects of an application's interaction with OVW are defined using the Application Registration File (ARF). The ARF provides OVW with important information, such as:

- How to integrate the application into the OVW menu structure.
- How to invoke the application based on the user's run-time selection of menu items or executable symbols.
- How to manage the application process.
- Where application-specific help information is located.
- Which fields to display in the OVW dialog boxes.

You can choose how tightly your application is integrated into the OVW user interface. At the simplest level, your application might be integrated only through the OVW menu structure. That is, you might provide an ARF that causes the application to appear as a menu item in the OVW menu structure.

Application Registration Files are stored in the directory `/usr/OV/registration/$LANG`. The `$LANG` environment variable provides support for native language localizability. If not otherwise defined, `$LANG` is assumed to be `C`, which means no native language support is provided. (See `lang(5)` for more information on native language localization.)

Application Block

Applications integrate into OVW using structures in the ARF called application blocks. Application blocks define many aspects of the interface between OVW and applications. For convenience, all registration file entries consistently use a C-like syntax.

The application block specifies, for a single application, the following behavior:

- How the application is incorporated into the OVW menu structure (if the application appears in the OVW menu bar).
- Instructions for invoking the application command.
- The version, description, and copyright strings displayed for the application.
- Where application help information is located.
- The format of dialog boxes that OVW can display on behalf of the application.

An ARF contains a single application block that represents a single application. An application can be defined only once, and its name must be unique. The application block has the syntax:

```
Application "<your app name>" {  
    ..  
    one or more application specifications  
    ..  
}
```

where **<your app name>** is replaced by the string by which OVW will refer to your application.

The remainder of this section describes application specifications in more detail.

Version and Application Description

At run-time, end-users can view an application's version string, description, and copyright by selecting the **Help** menu item from the menu bar, followed by the **On Versions** menu item. For this information to be displayed properly, however, it must be preconfigured in the ARF.

The version is specified as a string in a **version** section. The application's purpose is specified in a **Description** block. The copyright is defined in a **Copyright** block. The following example demonstrates:

```
Application "My app" {  
    ..  
    Version "1.01";  
    Description {  
        "Sample application description",  
        "goes here."  
    }  
    Copyright {  
        "(C) 1993 Hewlett-Packard Company"  
    }  
    ..  
}
```

The version string text must be a single line. The **Description** and **Copyright** blocks are composed of one or more text strings separated by commas.

Command

The **Command** section lists the complete command line used to invoke the application. Regardless of whether the user selects the application through a menu item or through an executable map symbol, this command will be used to invoke the program.

Commands are delimited by “” characters. Commands that are tty-based (that is, non-X Windows applications) must be run in an X terminal window (**xterm(1)** or **hp(1)**, for instance). The following example illustrates:

```
Application "editor" {  
    ..  
    Command "xterm -e /usr/bin/vi";    ..  
}
```

By being defined at the application block level, the command is considered *global*. In other words, the application command will be invoked the same way every time, even if the user selects the application in different ways.

If you do not want a global declaration, you can:

- override the global declaration with an action declaration
- omit the global declaration as long as you have command statements within action declarations

Refer to the **OVWRegIntro(5)** man page for more information.

As you will see later, commands can be defined on an individual, *per-action* basis. Applications can be invoked in slightly different ways (for example, with different arguments) depending on the particular user selection. Per-action commands will be discussed later in the “Invoking the Application” section.

HP OpenView Windows Shell Environment Variables

OVW provides a run-time environment for applications. This run-time environment allows applications to access values of OVW environment variables. Applications have access to a number of key items, including the menu items used to invoke the application and the object selection list. These environment variables can appear on the command line that invokes the application.

The following OVW environment variables are available for your use:

OVW Environment Variable	Purpose
OVwSelections	A string containing the selection names of the objects in the selection list when the application is invoked. The names are separated by blank characters.
OVwNumSelections	The number of selection names in the selection list when the application is invoked.
OVwSelection_n (n=1..9)	OVwSelection₁ is set to the selection name of the first object in the selection list, OVwSelection₂ is set to the second name, etc. These strings are returned in the order in which their associated objects were selected. The selection list is limited to nine elements.
OVwActionID	The name of the action by which the application is invoked. Actions are defined in the ARF and are described later.
OVwMenuItem	If the application is invoked through a menu item, this environment variable will contain the label of the menu item that caused the action. Menu items are described later.

This example demonstrates a command that is invoked with the first element in the selection list:

```
Application "Telnet" {  
    ..  
    Command "/usr/bin/X11/xterm -e /usr/bin/telnet $OVwSelection1";  
    ..  
}
```

Process Flags

OVW can manage your application in a number of ways. By default, OVW invokes your application at run-time when the user selects the appropriate menu item or executable symbol. Each selection by the user causes another instance of the application to be loaded and executed. You can change this behavior by adding special flags to your command entry in the application block. For example,

```
Application "My app" {  
    ..  
    Command -Initial -Shared -Restart "/usr/local/bin/My_app";  
    ..  
}
```

There are currently three flags defined: **Initial**, **Shared**, and **Restart**. The following table describes their use.

Process Flag	Purpose
Initial	Tells OVW to start the application when OVW is started. The application will be invoked by the Command line in the application block.
Shared	Tells OVW that only a single instance of this application command should be running at any time. Further, this applicaiton instance will be "shared", servicing multiple action requests.
Restart	Tells OVW to restart the application if it should ever exit. This flag is intended for applications that are required for normal OVW operation, such as those managing the semantics of OVW maps.

If you want to use more than one process management option, they should be combined in the same **Command** entry as shown in the example.

Namefield

A single OVW object can have multiple names, such as:

- A selection name (OVW's unique name for the object).
- An IP Hostname (for IP networks).
- A user-defined or application-defined name.

When a user selects an object on a map, the OVW selection list is built using the object's selection name. The **NameField** section lets you choose other name forms that are used to construct the selection list. You can, for example, request that only IP host names are used in the selection list.

An indirect benefit of using a **NameField** section is that it guarantees that actions or commands are only allowed if all objects in the selection list have that name field. If one or more objects do not have the appropriate name field, the command will not be allowed (for example, the menu item would be greyed-out).

You can use any of the preregistered OVW name fields in the **NameField** section, or you can create your own. (Field definitions are described later in this chapter.) For instance, the previous example demonstrated how to invoke telnet with an OVW shell environment variable. This would work well if the selection item was the same as the hostname and the host was capable of supporting telnet. It would be inappropriate, however, for a modem. By supplementing the **Command** section of the registration file with a **NameField** section, you can guarantee that the command is invoked with a list of selected items appropriately filtered. You can extend the telnet example by filtering out all selected items that do not have host names, as in the following example.

```
Application "Telnet" {  
    ..  
    Command "/usr/bin/X11/xterm -e /usr/bin/telnet $OVWSelection1";  
    NameField "IP Hostname";  
    ..  
}
```

In this example, the command would only be enabled if all objects in the selection list have an IP Hostname name field. The command would be invoked using the first object in the selection list as an argument.

You can supply multiple names, separated by commas, in the **NameField** statement. OVW will check the object's names against each name specified in the **NameField** statement. If multiple names are specified in the **NameField** statement, OVW will use the first name that is valid for the object.

The **NameField** is a convenient way to filter selections from the selection list. As you will see later, you can construct more sophisticated filters using selection rules, which are described in the "Menu Registration" section.

Menu Registration

A key feature of HP OpenView Windows (OVW) is its ability to integrate network and systems management applications with a common menu interface. You use the ARF to integrate applications into the OVW menu structure.

This section teaches you how to integrate applications into the OVW menu structure through the ARF.

OVW allows you to integrate your application either on the main menu bar or under a menu cascade. Regardless of where your application is integrated in the OVW menu structure, the tasks in registering your application are the same:

- Specify the menu item on the main menu bar under which your application will be integrated.
- Specify the relationship, or linkage, between the main menu bar item and the next menu item or pull-down menu.
- Continue defining menu items in cascading menus as desired.
- When you have reached the menu entry in which your application should be invoked, define an action block specifying how to invoke the application.

Here are the steps in more detail.

Defining the Menu Structure

Application integration begins in the main menu bar. A MenuBar block is defined that specifies the menu label on the main menu bar under which the application will eventually appear. You can specify an optional mnemonic character to allow keyboard selection by typing a single character. For example:

```
Application "My App" {  
    ..  
    MenuBar "Configure" _C {  
        ..  
    }  
    ..  
}
```

In the previous example, application integration begins under the **Configure** menu bar entry. The user can select this menu item by either using the mouse or by typing 'C' once the menu bar is activated for keyboard input.

If the **Configure** menu item does not exist then OVW creates it for you. If it already exists, then OVW will simply make the association to the preexisting menu item.

The main menu item on the menu bar is now defined. The next step is to define additional menu items and the application invocation. These are done through a menu statement defined within the MenuBar block. The menu statement specifies the menu label and its function. One of three different functions can be used:

Action	Purpose
f.menu	Provides the declaration of a menu cascade within a menu. Menus in the OVW menu structure will be extended to include the new entry, or, if one does not exist, a new cascading menu will be created.
f.action	Associates an application invocation, or action, to the menu item. When the user selects the menu item, the application is invoked appropriately. The application is notified of the selected action.
!	Provides a short-cut mechanism to integrate shell commands into menu selections. Applications invoked in this way are not notified of the user's selected action, nor can selection rules be used. (Selection rules are described later). If you desire these features, you should use the general f.action function instead.

The following example demonstrates these menu concepts. The details of invoking the application will be described later.

```

Application "My App"
{
  ..
  MenuBar "Monitor" _M {
    "Local Network" _L f.menu NetItem;
  }

  Menu NetItem {
    "Trends..." _T f.action Trends;
    "Statistics..." _S f.action Stats;
  }
  ..
}

```

In this example, the user could select the **Monitor** menu label from the main menu bar. A pull-down menu containing the **Local Network** menu item is then made available. Selecting the **Local Network** menu item causes a cascading menu to appear that contains two more choices, **Trends...** and **Statistics...** Selecting either of these menu items causes the appropriate applications to be invoked.

Notice the single character mnemonics in the menu statements in the example. Mnemonics are one of the optional modifiers for a menu statement. You can optionally specify

- A single character mnemonic for keyboard selection, as in the example.
- A precedence value indicating how the menu item should be ordered. Menu items are displayed according to the ordered precedence, and, within items of equal precedence, in the application name order. Precedence is specified as an unsigned integer value; 100 is the highest precedence, 0 is the lowest.
- A keystroke sequence that invokes the menu selection without displaying the menu. This is also called a keyboard accelerator, and is specified using the Motif keyboard binding syntax.

Note

When adding your application into the OVW menu structure, avoid creating new items in the OVW main menu bar. Instead reuse existing menu structure whenever possible. Also, avoid creating menu cascades more than three levels deep.

Invoking the Application

Once the menu structure is in place, the last step is to define how the application is invoked.

Application invocation is defined using an Action block. The Action block specifies how the application command is invoked, how the process is managed, and how the application uses the selection list from the map.

The following definitions can be used in an Action block:

Action Block Entries	Purpose
SelectionRule	Provides a mechanism to guarantee that the application is invoked only if the selected objects meet predefined criteria. The selection rule is a logical expression involving capability fields. The logical expression can use logical AND (&&), logical OR (), or logical NOT (!) operators.
MaxSelected	Specifies the maximum number of objects that can be selected on the map for the action to be enabled. If not specified, any number of objects may be selected for the action. If set to 0, no objects may be selected.

Action Block Entries	Purpose
MinSelected	Specifies the minimum number of objects that must be selected on the map for the action to be enabled. If MinSelected is set to some value, then the selection list must contain at least that number of objects. If MinSelected is not set, then any number of objects can be selected. (If MinSelected is not set and a SelectionRule is set, the selection list must contain at least one object. If MinSelected is set to 0 and a SelectionRule is set, the selection list can contain 0 or more objects.)
Command	Specifies the command line used to invoke the application on a per-action basis. Commands specified in an Action block override commands specified globally in the Application block. If no command is specified globally, a command must be specified for each action defined.
Process Flags	You may define specific process management instructions on a per-action basis. The Initial , Shared , and Restart flags may be used on a per-application basis and override any global process flags.
NameField	The NameField entry provides the same object name field validation as described in the Application section, but on a per-action basis.
CallbackArgs	Using the OVW API, you can designate a callback routine that is invoked when the user selects this action. Using the CallbackArgs entry, you can specify a string containing arguments which are passed to the callback routine.

The following example demonstrates an action block specification:

```
Application "My App"
{
  ..
  MenuBar "Monitor" _M {
    "Network Activity" _A f.menu "Net Activity";
  }
  Menu "Net Activity" {
    "Trends..." _T f.action Trends;
    "Statistics..." _S f.action Stats;
  }
  Action Trends {
    SelectionRule isNode && isDevice;
    MinSelected 1;
    MaxSelected 1;
    Command "/usr/local/bin/myapp ${OVwSelection1}";
    CallbackArgs "-trends -verbose";
  }
  ..
}
```

In this example, the user can select the **Trends...** menu item from the menu cascades to request that the application be invoked. OVW only executes the application if there is a single object selected on the map, and that object has the **isNode** and **isDevice** capabilities. If these qualifications are met, the application is invoked with the name of the selected item. The **CallbackArgs** statement defines a string that is passed to the application's callback routine when the user invokes this action.

This completes the discussion of the Application Registration File. The next major section addresses the second form of registration file, the Symbol Type Registration File.

Symbol Type Registration Files

Symbols are graphical representation of objects in HP OpenView Windows (OVW). Symbols can either be icon symbols or connection symbols. Icon symbols represent network or system management elements, while connection symbols represent connections between elements.

Each symbol is identified by its symbol type. The symbol type is defined by a symbol class/subclass pair. The symbol class defines the symbol category, while the symbol subclass defines a particular element within that subclass.

OVW provides you with a rich set of predefined symbol classes and subclasses. Figure B-1 shows some of the symbol classes and subclasses provided with OVW.

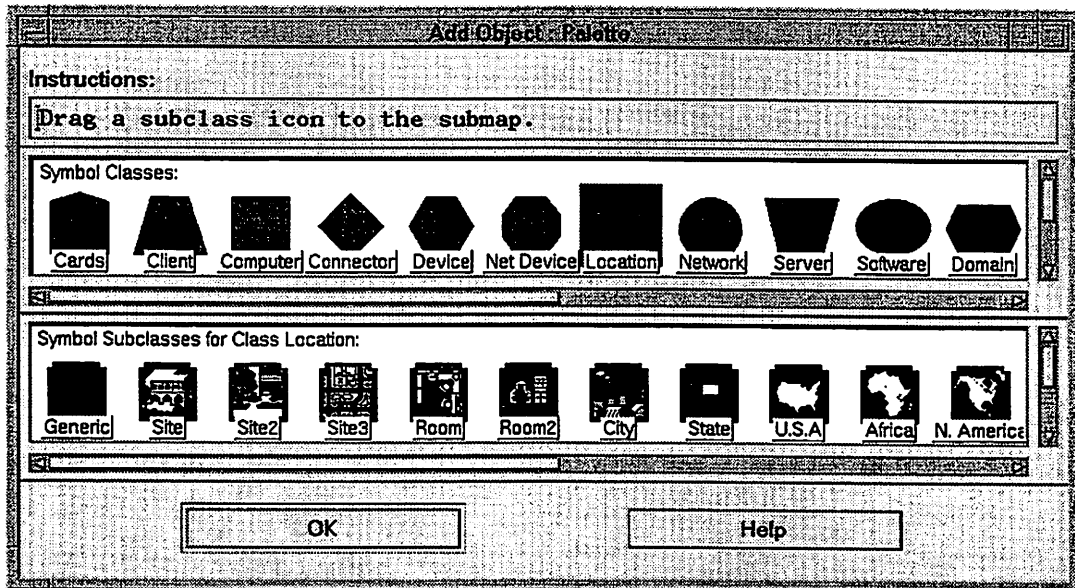


Figure B-1. OVW Symbol Palette

Symbol Type Registration Files are used to define symbol classes and subclasses in OVW. The predefined OVW symbol classes and subclasses are themselves defined using various symbol type registration files in the `/usr/ov/symbols/$LANG` directory. You are encouraged to use these predefined symbol classes and subclasses in your applications.

You may find, however, that existing classes and subclasses are not adequate for your needs. You can add new symbol subclasses to existing symbol classes or you can define your own symbol classes. This portion of the chapter shows how to define symbol classes and subclasses. Only icon symbol classes and subclasses are described here, as they are the most commonly used symbol variety. If you need to define connection symbols, see the `OVwRegIntro(5)` man page. (All references to the terms “symbol classes” and “symbol subclasses” throughout the remainder of this section refer to icon symbol classes and subclasses.)

Defining Icon Symbol Classes

An icon symbol class is represented graphically by its external shape. Symbol classes are defined using files in the `/usr/ov/symbols/$LANG` directory. Each file in the `symbols` directory contains one or more symbol class definitions, where each definition is composed of three elements:

- The class name (a string).
- The description of the external shape (a series of line segments or arcs).
- An optional integer scale factor.

Complete examples of symbol class definitions are provided in the following sections. Symbol class definitions based on line segments are treated first, followed by class definitions based on arcs.

Defining Classes Using Line Segments

Line segments are often used to define most icon symbol classes in OVW. As shown in figure B-2, the majority of icon symbol classes provided with OVW are defined using line segments.

Shapes are defined using a series of points that lie within a conceptual square. Points are specified using integer values which, by default, may range from -1 to 1 on both the X and Y axes. The conceptual square has center coordinates of (0,0).

The following example shows the OVW 'Computer' symbol class and the symbol class definition that produced it.

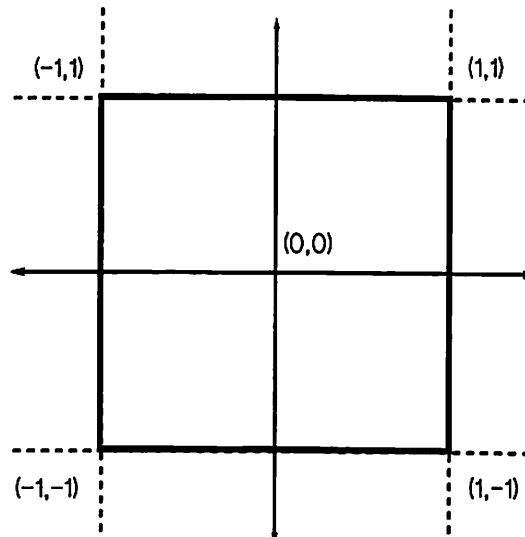


Figure B-2. "Computer" Symbol Class

```
SymbolClass "Computer" {  
    Segment (-1,-1) to (1,-1) to (1,1) to  
        (-1, 1) to (-1,-1);  
}
```

By default, the scale is 1. In other words, the integer point values may range from -1 to 1 on both the X and Y axes. A scale of 1 limits you, though, to simple, uninteresting shapes.

You can create more complex shapes by increasing the scale and, therefore, using more points. OVW will interpret your points relative to your scale and will display your shape in a standard size. The following example demonstrates how to use the scale entry.

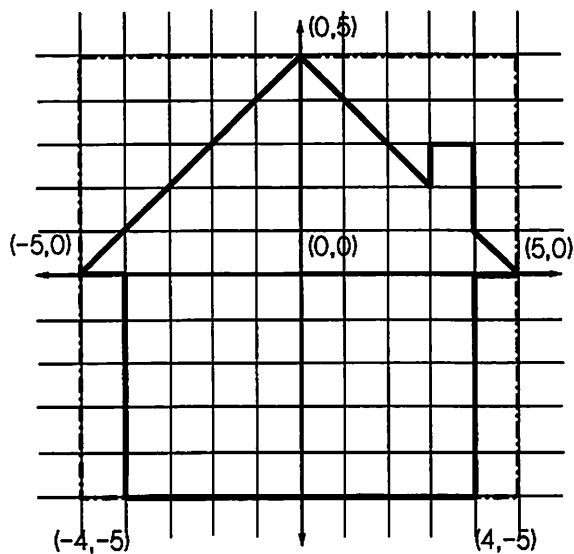


Figure B-3. "House" Symbol Class

```
SymbolClass "House" {
    Scale 5;
    Segment (-4,0) to (-5,0) to (0,5) to (3,2) to
        (3,3) to (4,3) to (4,1) to (5,0) to
        (4,0) to (4,-5) to (-4,-5);
}
```

Note that the example segment definition is not closed explicitly. OVW completes the definition by drawing a line from the last point to the starting point.

You have much freedom in defining symbol class shapes. There are, however, some aspects of shape definition you should be aware of:

- Specifying points wholly within the square may not produce a symbol class shape as large as desired. You can change the actual symbol class icon size by specifying coordinate values that exceed your scale. This has the effect of making your symbol class icon larger. Your point values should not exceed 150% of scale, though, or symbols may overlap when OVW places them on the map.

- The point (0,0) should be contained within your shape definition. If it is not, OVW cannot correctly connect symbol classes on the submap.

Defining Classes Using Arcs

As an alternative to using line segments, you may also define icon symbol classes using arcs. Arcs allow you to easily define such class shapes as circles, ellipses, and wedges that would be tedious to define using line segments.

The definition of an arc is composed of three components:

Keyword	Purpose
Size	This is the width and height of a conceptual rectangle in which the line will rotate. If the width and height are equal, the arc will have a constant radius. Elliptical arcs may be achieved using height and width values that are unequal.
Origin	Specifies an origin for rotation within the conceptual rectangle. The center of the conceptual rectangle is assumed to have the coordinates (0,0).
Rotation	This specifies the starting angle and number of degrees of rotation. A starting angle of 0 degrees is equated to a three-o'clock position, and rotation occurs in a clockwise direction.

The following example demonstrates how to define a symbol class with a circular shape.

```
SymbolClass "Network" {
    Scale 2;
    Arc Origin(0,0) Size(2,2) Rotation 0, 360;
}
```

If you need further information about defining symbol classes using arcs, see the **OVWRegIntro(5)** man page.

Note Arcs and Segments cannot be combined in a single shape definition.

Optional Class Specifications

There are a few optional class specifications you can use when you define symbol classes.

■ Default Layout.

When end-users double-click on a symbol on a map, the symbol may explode into a submap. The **DefaultLayout** entry specifies the default layout algorithm for the child submap of an object represented by a symbol with this symbol class. (This default can be overridden by an end-user or application when the submap is created.) The choices are: **Ring**, **Bus**, **Star**, **PointToPoint**, and **RowColumn** (the default). An example demonstrating default layout is provided later.

■ Default Status Source.

A symbol can get status from a variety of sources:

- From the underlying object (object status).
- From the parent object of a child submap (compound status).
- From the symbol itself (symbol status).

The **DefaultStatusSource** entry sets the default status source for symbols of this class. If not set, symbol classes have object status.

You can assign default status source to symbol classes by defining a **DefaultStatusSource** structure in the class definition. You need to supply one of three status source values: **Object**, **Symbol**, or **Compound**. The following example shows how to set the default layout and the default status source for a symbol class.

```
SymbolClass "Computer" {  
    Segment (-1,-1) to (1,-1) to (1,1) to (-1,1);  
    DefaultLayout RowColumn;  
    DefaultStatusSource Compound;  
}
```

■ Default Capabilities.

An object can be added to the map by selecting a symbol type on the symbol palette and dragging it to a submap. OVW automatically creates an object that is associated with the symbol. By default, the new object has no capabilities.

You can assign default capabilities to objects by defining a **Capabilities** statement in your class definition. Whenever a symbol is added to a map, OVW will use the symbol's default capabilities to initialize the capabilities of the associated object being added. For example,

```
SymbolClass "Computer"
{
    Segment (-1,-1) to (1,-1) to (1,1) to
           (-1, 1) to (-1,-1);
    Capabilities {
        isNode = 1;
    }
}
```

■ Variety.

There are actually two forms of symbol classes: icons and connections.

Icons have already been discussed.

Connections are symbol classes that connect other symbols. You can specify that a class is a connection by using the **Variety** definition and specifying that the class has the value **Connection**. (The default is **Icon**).

The `/usr/ov/fields/C/Connection` file has some examples of defining connection symbol subclasses:

```
SymbolClass "Connection"
{
    Variety Connection;
    Capabilities {
        isConnection = 1;
    }
}

SymbolType "Connection" : "Dashed"
{
    Linestyle Dash;
    LineDashPattern 10, 5, 10, 5, 10, 5;
}

SymbolType "Connection" : "Dotted"
{
    Linestyle Dash;
    LineDashPattern 2, 2, 2, 2, 2, 2;
}
```

```

SymbolType "Connection" : "DotDash"
{
    LineStyle Dash;
    LineDashPattern 3, 2, 3, 5, 3, 2, 3, 5;
}

```

In the previous example, the class specification is defined first.

Note

For the HP OpenView SNMP Management Platform release 3.x product, the specification of the **Variety** for **Connection** should only be made for the "Connection" class. This is done in the **ovw** registration file, but the parser will handle multiple instances of this specification.

For every **SymbolType** in the **Connection** class, **LineStyle** must specify either **Solid** or **Dash**.

- The "Generic" symbol type (auto created) for the **Connection** class is a solid line.

When **LineStyle** is **Dash**, the additional specification of **LineDashPattern** is needed.

LineDashPattern must specify two parameters: *draw_length* and *gap_length*. These parameters indicate a repeating pattern of lines and spaces. The whole pattern repeats as needed, and there is no limit to the number of parameters patterns specified. The length is in terms of pixels.

For further information about connection symbol classes, see the **OVwRegIntro(5)** man page.

Defining Symbol Subclasses

A Symbol Subclass definition allows you to define a new symbol subclass within a class of symbols. Subclasses are represented using bitmaps that are superimposed on symbol class shapes. You can use any standard bitmap editing tool, such as `bitmap(1)`, to create subclass bitmaps.

Symbol subclasses are defined using a symbol type block. The symbol type block specifies the name of the subclass and the class to which it belongs. The symbol type can be considered the combination of the symbol class and the symbol subclass. It has the following syntax:

```
SymbolType "class name": "subclass name" {  
    ...  
    subclass specifications  
    ...  
}
```

The “class name” is the name of an existing symbol class. The “subclass name” is a unique name for the new subclass. The subclass specifications define the bitmaps that are used for the subclass as well as other characteristics of symbol subclass behavior. Some subclass specifications are required, while others are optional.

Required Subclass Specifications

Two subclass specifications must be present in every icon symbol subclass definition:

- **Filebase.**

This entry defines the basename for a symbol subclass bitmap. A bitmap definition is composed of two parts: `filebase.size.p` (the bitmap) and `filebase.size.m` (the bitmap mask). You should provide a pair of bitmap/bitmap mask files for each bitmap size you support. (You must provide at least one bitmap pair.)

OVW uses different bitmap sizes whenever the dimensions of an OVW window changes. OVW attempts to “scale” the window contents appropriately, which may mean choosing a different symbol bitmap to optimally fit the window scale. If the appropriate bitmap file does not exist, OVW will choose the closest possible match. HP recommends that you provide bitmaps in the following sizes (in pixels): `20x20`, `26x26`, `32x32`, and `38x38`. It is mandatory that at least a `32x32` or `38x38` bitmap be provided.

- **Cursor Size.**

Whenever the end user selects a symbol type from the symbol palette and moves the symbol type to a map, the cursor shape changes from a pointer to a bitmap. The `CursorSize` entry defines the size of bitmap to be used as the cursor. HP recommends a cursor size of `38x38` pixels.

The following example demonstrates both the Filebase and CursorSize entries:

```
SymbolType "Software": "Application" {  
    Filebase "app2";  
    CursorSize 38;  
}
```

In this example, the cursor will be 38 by 38 pixels and will be based on the bitmap formed by the files **app2.38.p** and **app2.38.m**.

Optional Subclass Specifications

Additionally, there are optional subclass specifications you can use in your symbol type definitions. These specifications change the basic behavior of the symbol type:

- **Default Layout.**
This specification was covered with symbol classes. A **DefaultLayout** entry in a symbol subclass definition overrides any default layouts specified in the class definition.
- **Default Status Source.**
This specification was also covered with symbol classes. A default status source specification in the subclass definition overrides any default status source statements in the symbol class definition.
- **Default Capabilities.**
This specification was covered with symbol classes. A capability defined in a symbol class is inherited in the symbol subclass. If you specify a subclass capability specification with the same capability field as in the class definition, the subclass definition overrides the class definition.

For further symbol subclass examples, consult the registration files provided with OVW (in the **/usr/OV/symbols/\$LANG** directory).

Field Registration File

Fields are the building blocks used to construct objects. Fields define individual characteristics, which, when grouped together into a single entity, form an object. This section describes how to use the Field Registration File (FRF) to define fields for objects. If your application does not use or define HP OpenView Windows (OVW) objects, you can skip this section.

Fields are defined using a Field Registration File. Field Registration Files are stored in the `/usr/OV/fields/$LANG` directory.

Defining Fields

There are three components to a field definition:

- The field name.
- The data type of the field.
- Flags that indicate how the field is used.

Every field must have a data type and a unique name. Flags can be optionally supplied to alter how the field is treated by OVW.

Use the field block structure to define fields. The FRF contains one or more field blocks, where each field block defines a single field. The field block has the syntax:

```
Field "field name" {  
    Type "field type";  
    [Enumeration <field enumeration>+;]  
    [Flags "field flags";]  
}
```

The field name and field type are required.

Field types and flags are described next.

Field Type

Every field has an associated data type. The **Type** entry declares the field's data type. **Type** is a required entry, and must be set to one of four possible types:

- **Boolean.** This type can only have the values True or False.
- **String.** A standard character string, limited to 256 characters in length.
- **Enumeration.** Declares the field to be an enumerated type. All possible values for the enumerated type are declared in a separate **Enumeration** statement. An example enumeration appears later.
- **Integer32.** A 32-bit signed integer.

Field Flags

While the **Type** field specifies the data type of the field, the **Flags** field specifies how the field is treated by OVW. OVW gives you the flexibility to treat fields in different ways. For instance, you can specify that a particular field is a “name” field, which means that the value of the field is unique across all objects that contain the field. You can also specify that a particular field should appear in the **Locate: By Attribute...** dialog box on the OVW menu interface. These behaviors, as well as others, are defined using the **Flags** statement.

There are five types of flags (behavior) that can be applied to fields:

- **List.** Specifies that the field contains a list of fields of the same type, rather than just a single field. The **list** flag allows you to associate multiple values of the same type to a single field definition. The **list** flag is only valid for fields of type string.

For example, you might define a field for the names of administrators of your computer network. The type would be **string**, and the flag field would be set to **list** to allow you to store multiple names.

- **Name.** Indicates that the value of the field uniquely identifies an object containing this field definition. This flag can only be used for fields of type **string**.

For example, the **IP Hostname** field is defined using the **name** flag, since each node on a network must have a unique IP Hostname.

Non-string data types with unique values (such as a network adapter's link-level address) can serve as name fields provided that they are converted to and stored as strings.

Note that the values given to name fields must be unique; no two objects can have the same value for a name field.

- **Locate.** By specifying this flag in the field definition, the field name shows up in the **Locate: By Attribute...** dialog box when end-users attempt to locate an object.

You should use care in setting this field. If you do not specify the **locate** flag, end-users won't be able to locate an object based on your field. On the other hand, indiscriminate use of the **locate** field will result in an overabundance of locate entries, making that dialog box harder to use effectively. Only set this flag if end-users have a strong need to locate objects through this field. This flag cannot be set together with the **list** flag.

- **Capability.** By setting this flag, the field is considered by OVW to be an object capability. In other words, the field is used to classify an object. For example, OVW defines fields such as **isRouter** and **isDevice**. These fields are used to classify objects and are often used in assertions. The **capability** flag may only be used for boolean and enumerated field types. Capability fields determine menu greying (which menu operations are available based on the selected objects).
- **General.** Fields with this flag appear in a special General Attributes dialog box associated with every object. This is for fields that are not application-specific and do not appear in any application-specific dialog box. The **vendor** field is a good example of a general field.

As you can see from the flag descriptions, not all data types can be used with all flags. The following table summarizes valid data type/flag combinations.

Flag	Data Type			
	Integer	String	Boolean	Enumeration
List		X		
Name		X		
Capability			X	X
Locate	X	X	X	X
General	X	X	X	X

Example Field Definitions

This section shows how to use field types and flags. The first set of examples demonstrates the use of **capability** fields.

```
Field "isNode" {
    Type boolean;
    Flags capability;
}

Field "isDevice" {
    Type boolean;
    Flags capability;
}

Field "isComputer" {
    Type boolean;
    Flags capability;
}
```

You can see that these three fields are used to determine an object's capabilities. An object could have device, computer, or node capabilities. In fact, an object could even have a combination of these capabilities. Capability fields are used to classify objects in OVW and, therefore, cannot be used to uniquely identify (or name) an object.

The next example shows the syntax for defining enumerated types and using the **locate** flag. (To see a procedure showing how you define the actual values, refer to the "Adding Values for Vender and SNMP Agent Fields" section in chapter 3.

```
Field "IP status" {
    Type enumerated;
    Enumeration "Unset",
               "Unknown",
               "Normal",
               "Marginal",
               "Critical"
               "Unmanaged" ;
    Flags locate;
}
```

This example shows how you might use enumerated types to classify the status of nodes on a network. Each node on a network would have a status value from this set. By setting the **locate** flag, end-users have the ability to search for, or locate, all nodes on the network with a particular status. It would not make sense to set the **capability** flag or **name** flag for this field.

The next example demonstrates how to use a field as a **name** field. Since the host name of a computer would be unique for all nodes on the network, the **name** flag can be set for the field.

```
Field "IP Hostname" {  
    Type string;  
    Flags name;  
}
```

The following example demonstrates how to use the **list** flag in field definitions.

```
Field "Service Contacts" {  
    Type string;  
    Flags list;  
}
```

This field definition specifies that the field's value is actually a list of strings, such as the names of people who can service equipment.

Combinations of Flags

Flags can be applied to different data types and for different purposes. The examples above do not use combinations of flags, but there may be occasions where you want to use flags in combination. Flags can be used in the following ways:

- All flags can be used individually.
- The **name** and **locate** flags can be used together.
- The **capability** and **locate** flags can be used together.
- The **general** flag can be used with the other valid combinations described here.

If a combination is not listed here then it is not permitted. For example, you cannot combine the **name**, **capability**, and **locate** flags in a single flags statement.

How Registration Files Are Processed

The Application, Symbol Type, and Field Registration Files are processed in different ways. The Application and Symbol Type Registration Files are read once each time HP OpenView Windows (OVW) is started. When OVW begins initialization, it searches the various pre-defined directories for registration files. If any files are found in those directories, the files are assumed to be registration files. For every Application or Symbol Type Registration File found, OVW executes these steps:

- Opens the file.
- Parses it for correctness.
- If an entry is found to be in error, OVW prints error information to stderr, indicating the file and line where the error occurred. Possible errors include:
 - Duplicate entries.
 - Ill-formed entries.
- If a valid entry is found, OVW performs the appropriate operation (for example, adding a menu item to the OVW menu structure or adding a symbol to the OVW symbol palette).

Field Registration Files are processed differently than the other registration files. Field Registration Files are not processed when OVW is started. Rather, administrators manually add the field entries to the OVW object database using a special command. System-administrators execute the special **ovw -fields** command to add fields to the object database. The Field Registration Files are parsed for correctness and checked for errors just as the other registration files, and errors are reported to stderr. See the **ovw(1)** and **OVWRegIntro(5)** man pages for more information about processing of Field Registration Files.

Summary

The Application, Symbol Type, and Field Registration Files provide a static mechanism to configure HP OpenView Windows (OVW) behavior.

Application Registration Files are used to integrate applications into the OVW user interface. Application Registration Files define all aspects of an application's interaction with OVW, including how applications are added to the OVW menu structure, how applications are invoked and application processes managed, and where application-specific help information is located.

Symbol Type Registration Files are used to define symbol classes and symbol subclasses. Symbol classes can represent either icon or connection symbols. Icon symbol classes are defined using line segments or arcs, while icon symbol subclasses are defined by standard bitmaps. Connection symbol classes and subclasses are described in the man pages. OVW provides you with a large set of predefined symbol classes and subclasses, but you can define your own as well.

Field Registration Files are used to define fields, which are attributes of objects. Field Registration Files define the data type and behavior for fields.

Reference

This appendix contains the Reference Pages (man pages) for the HP OpenView Network Node Manager product.

NAME

findroute - determine IP routing using SNMP requests

SYNOPSIS

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

DESCRIPTION

findroute determines the route taken by IP packets between two nodes by using the Simple Network Management Protocol (SNMP). An iterative process is applied in which the IP routing table of the *source* node is searched via 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, *findroute* prints the routing node name, the routing node's IP address, the next node in the route, and the next node's IP address. Names are determined from IP addresses as described in *hosts*(4).

The graphical interface equivalent to *findroute* is the Locate Route operation under the Diagnose -> Network Connectivity menu; execute the graphical interface using *ovw*(1).

Options

findroute recognizes the following options:

- n Do not truncate host names. By default, output host names are truncated to 20 characters.
- c *community* Override the SNMP community name (as configured in */usr/OV/conf/ovsnmp.conf*) to *community*. See *ovsnmp.conf*(4).

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *findroute* behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

WARNINGS

findroute 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.

AUTHOR

findroute was developed by HP.

FILES

/usr/OV/conf/ovsnmp.conf SNMP configuration

SEE ALSO

rnetstat(1), *ovw*(1), *snmpd*(1M), *ovsnmp.conf*(4).

*/usr/OV/doc/** (for relate RFS's)

NAME

ipmap - display map of network discovered by netmon using HP Openview Windows

SYNOPSIS

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

DESCRIPTION

IPMap is an application that is integrated under OVW. It reads IP topology data from *ovtopmd*(1M) 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 starts its synchronization phase. While *IPMap* is synchronizing *OVW* displays a 'Synchronizing' message on the status line of the displayed submaps. During synchronization, *IPMap* brings the map up to date with the *ovtopmd* database by creating symbols for new network objects. During this time, *IPMap* is unable to respond to some *OVW* requests, and some functions are not accessible by the user. For example, *OVW* restricts users from deleting symbols and objects while any application is synchronizing. Also, *IPMap* will not appear 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 that operation until it is through with this phase. That is *OVW* will manage the objects as the user requests but the *ovwManageObjects* event sent to *IPMap* is queued by *IPMap* until it is through with synchronization. Once *IPMap* is through with synchronization it enters a mode where it handles incoming *OVW* events and SNMP traps from *ovtopmd*(1M).

IPMap will also display the 'Synchronizing' message during operations that can take a long time. This includes configuration changes, delete operations, manage and unmanage operations. The same restrictions are placed on the user during these intermittent synchronization phases. However, *IPMap* will appear in the list of applications in the add dialog boxes. If the user attempts to add an object while *IPMap* is synchronizing *IPMap* will not respond to *OVW* events until it is through synchronizing.

The user may 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 be deleted and readed as *netmon* moves them from one segment to another.

User Editing of Map

IPMap supports map edits by the user. This is accomplished through the *OVW* user interface. There are several ways in which the user can modify the map through the user interface including: add symbols, delete symbols and cut/paste 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 via different menu items in *OVW*. Each submap type has a different set of symbol types that it allows the user to add. This is explained in detail in the submap sections below.

IPMap supports the deletion of symbols from the map. There are two cases where *IPMap* will not allow a symbol to be deleted. The internet symbol that is placed in the root submap can not be deleted. Similarly, certain interfaces discovered by *netmon* can not be deleted. All other symbols may be deleted by the user.

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 all maps. Therefore if the user wishes to delete an object so that it is rediscovered by *netmon* (a network for example) he must delete that object from all maps.

Finally, cut and paste is supported by *IPMap* in the internet submaps and segment submaps. The details of these operations are covered in the submap sections below. Although the cut and paste operations can be accessed in all of the submaps (i.e. *OVW* allows the user to perform these operations) *IPMap* will only respond to the cut/paste on the internet submaps. In other cases, the symbol remains in the user plane after the paste.

Options

-b number of symbols to be batched for creation. By default this value is 15 but it can be changed to either improve performance or improve user feedback. By setting this value to a smaller number the

user will see smaller, more frequent updates to the map when symbols are added. This benefit causes a decrease in performance, however, since it requires more context switches. By setting this value to a larger number the user will see larger, less frequent updates. As a consequence, large maps can be drawn faster.

- d disable *IPMap*. *IPMap* is enabled by default. By using this option *IPMap* is disabled. When *IPMap* is disabled it is executing but is dormant. It will not update the map with any topology data. The user has the option of creating a map in which *IPMap* is enabled by changing the appropriate field in the *IPMap* configuration dialog box when the map is created.
- s configure status defaults. The valid values are on, off, connStatusOn, connStatusOff and compoundNodeOn.

Value	Meaning
on	set the status of interfaces
off	do not set any IP status
connStatusOn	connection status set the same as underlying interface
connStatusOff	connection status is not set
compoundNodeOn	status of node symbols reflect the status of all components

- D demand creation level for submaps. By default *IPMap* is configured to place all submaps and symbols in the map as soon as they're 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 wish to reduce the amount of system resources consumed by *OVW* and *IPMap* 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.

Value	Meaning
3	Only create the internet submap initially.
2	Create the internet submap and all network submaps.
1	Create the internet, network and all segment submaps.
0	Create all submaps.

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 " (*)" but this can be changed by using the -l and -L options explained below.

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

If the submap demand level is set to 3 (create only the internet submap) then the user may not successfully partition the internet submap using cut and paste as described below in the *Internet Submap* section. 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 (and therefore have a submap created below them).

If the submap demand level is set to 1 (create submaps down to the segment level) then the user may 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 since *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 since *IPMap* will not create a child submap for an interface.
- t turn on detailed tracing. See the Tracing section below.

These options can be set in the *IPMap* registration file on the command line.

IPMap Configuration

IPMap can be configured by the user by using the application configuration dialog box in the *OVW* menu. There are four items that are configurable in this box.

First is "IPMap enabled for this map?" This indicates if *IPMap* should create symbols on this map or not. Press TRUE if *IPMap* symbols are desired. Press FALSE if the user does not want *IPMap* to create any symbols on this map. This option can only be accessed when a map is being created. After the map is created, this option is grayed out. When this option is set to FALSE for a map, *IPMap* will still run but it will not update the map. If a new map is opened where *IPMap* is enabled then it will begin updating that map.

The second option is "Should IP status be shown on this map?" This indicates if *IPMap* should set the status of objects on the map or not. Press TRUE if *IPMap* should set the status. Press FALSE if you do not want any IP status on this map. This can be changed at any time.

The third item is "Should status of interfaces be shown on connections?" This indicates if connection lines show interface status. If TRUE is pressed then the status of connection lines are set to be the same as the interface that connection represents. If FALSE is pressed, then connections will always show a normal status. This can be changed at any time.

The forth item is "Should status of nodes be IP only?" This indicates if *IPMap* should explicitly set the status of node symbols on the map. If set to TRUE then *IPMAP* will set the status of nodes symbols based on the IP context in which a node symbols resides. For example, if a gateway has two interfaces. One interface on network A and the other on network B. If the interface on network A is down, the node symbols for that 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 that gateway in the context of network B remain in normal status. The gateway symbol on the internet submap becomes marginal since it is not in either context. The status of non-IP component symbols in the node submap are not considered for status propagation. If this option is set to FALSE then the status of all components of a node a propagated to the node symbol according to the propagation rules set for the map. In the case of the example above the status of all symbols for this gateway will 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 that gateway's interface that is attached to that network. This facilitates the display of interface status on higher levels of the map. When a gateway's interface goes down it is very easy to determine which network the bad interface is connected to. This is configurable, see the *IPMap Configuration* section.

One case where a connection is not a symbol of an interface object is the case of a serial link between gateways. See the *Serial Network* section 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 are allowed to be added 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 breath to the default *IPMap* submap hierarchy. This is particularly useful in cases where the default internet submap has many symbols or in cases where the user wants the map to represent some logical or physical hierarchy (geographical regions or administrative domains). It is accomplished by creating new internets and using cut/paste to move symbols from the original internet submap to the new ones.

The first step is to create the new internets. This is done by adding either an internet symbol (in the network class) or a symbol in the location class to the internet submap. The user must wait for *IPMap* to complete its synchronizatoion phase before adding these symbols. Once 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 it happens that there are multiple connections between two symbols, *OVW* will automatically create a compound connection submap showing all of the connections between the symbols.

When new gateways or networks are discovered by *netmon* *IPMap* will add them to the original internet submap. It is then the user's responsibility to move them to the appropriate internet submap. If the user, however, wishes to add a gateway or network he can 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, *IPMap* 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 *ovtopmd*.

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 *OVW*. 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 partition a network into segments manually (this 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. *IPMap* will check to make sure 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 then *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 *netmon*. Other applications, however, can add whatever components to the node that they wish.

The only symbol recognized by *IPMap* in this submap is the IP interface in the card class.

Serial Networks

Serial networks are treated as a special case by *IPMap*. A serial network will be displayed in one of two ways. First, 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 then 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 then 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 on without effecting the behavior of the IP topology or *ipmap*. See *OVwRegIntro(5)* for more details on *ovw(1)* 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*(1M) 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.sysObjectId** associated with the node, as determined via the *oid_to_sym*(4) configuration file. For nodes without a **sysObjectId**, the default symbol type is determined based on the field attributes associated with an object, and will be one of the following:

Connector:Gateway
Connector:Bridge
Connector:Multiport
Connector:Repeater
Computer:Generic

Nodes that have no particular IP topology behaviors and no SNMP **sysObjectId** will have a default symbol type of **Computer:Generic**. See *ovtopmd*(1M) for more discussion on the attributes associated with nodes.

In addition to the above methods for determining a node symbol type, the default symbol type for a node may be based on the OpenView Network Node Manager release 2.0 object type. This type is saved during the conversion process by *convertnnm*(1M), and converted to an *ovw*(1) symbol type via the *nnm_to_ovw*(4) configuration file. This method will override any other determination of symbol type.

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**.

TRACING

IPMap has three levels of tracing. You can trace traps from netmon, events from OVW and other miscellaneous internal items. You turn on tracing using these nettl commands as root:

```
nettl -e OVW -tracemon <mask> -f /usr/adm/nettl
nettl -e OVWEXTERNAL -tracemon <mask> -f /usr/adm/nettl
```

where <mask> is the bit mask for the type of tracing that is desired. *IPMap* supports 3 different masks:

Trace Type	Mask
Netmon Traps	0x00001000
OVW Events	0x00002000
Miscellaneous	0x00004000

By default *ipmap* produces an abbreviated trace message for each trap or event that contains minimal information. If detailed tracing is desired, there are two ways to turn that on. First, you could use the **-t** option on the command line. Another way to do this is to set the variable *IPMAP_DETAILED_TRACE* in your environment before starting OVW. Either of these will cause *IPMap* to produce very detailed trace

messages containing the information found in the incoming traps and events.

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 *ovwdb*(1M) object database. These fields are described in *ovtopmd*(1M).

Submap Type Values

The *OVwSubmapInfo* structure (see *ovw.h*) contains a field named *submap_type*. For submaps created by *IPMap* this field is set to one of the following values:

Submap Type	Field Value
Internet Submap	2
Network Submap	3
Segment Submap	4
Node Submap	5

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages appear and the types of symbols. If *LANG* is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of *LANG*. If any internationalization variable contains an invalid setting, *ipmap* behaves as if all internationalization variables are set to "C." See *environ*(5).

IPMAP_DETAILED_TRACE enables detailed tracing in *IPMap*. See the Tracing section above for more information.

IPMAP_RESET_CLOSE_TIME is the time from which *IPMap* should get changes from the topology to update the map. If, for example, the closed time of a map is incorrect you can export this environment variable before starting *ovw*. It uses the same format as the *touch* command: *[cc[yy]]mmddhhmm[.ss]*.

International Code Set Support

Supports 8-bit and 16-bit character code sets.

FILES

/usr/OV/conf/\$LANG/oid_to_sym SNMP sysObjectId to symbol type mapping

AUTHOR

IPMap was developed by HP.

SEE ALSO

ovw(1), *ovwdb*(1M), *ovtopmd*(1M), *oid_to_sym*(4), *netmon*(1M), *nnm_to_ovw*(4), *OVwRegIntro*(5).

NAME

nmdemandpoll - poll a node for information

SYNOPSIS

nmdemandpoll *node* ...

DESCRIPTION

nmdemandpoll sends a request to *netmon(1M)* for it to poll the specified *nodes* in sequence. The *node* name must exist in the database; the *node* name corresponds to the Selection Name of the object in the database. If the Domain Name Service is in use, the Selection Name is usually the fully-qualified domain name of the device.

nmdemandpoll polls for 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 interface equivalent to *nmdemandpoll* is the **Diagnose → Network Connectivity → Demand poll** operation (see *ovw(1)*).

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang(5)*) is used instead of LANG. If any internationalization variable contains an invalid setting, *nmdemandpoll* behaves as if all internationalization variables are set to "C." See *environ(5)*.

International Code Set Support

Supports single-byte character code sets.

NOTES

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

netmon(1M) limits the number of simultaneous polling operations; therefore, *nmdemandpoll* may have to wait until outstanding requests are completed.

AUTHOR

nmdemandpoll was developed by HP.

FILES

/usr/OV/tmp/demand.* Temporary files for communicating with *netmon(1M)*.

SEE ALSO

ovw(1), *netmon(1M)*.

NAME

nmpolling - topology/status polling configuration

SYNOPSIS

nmpolling

DESCRIPTION

nmpolling is an OpenView Windows application that enables users to configure certain *netmon*(1m) polling intervals.

nmpolling registers in the *ovw* menu bar under the **Options: Topology/Status Polling -> IP...** menu item. When this item is selected, *nmpolling* presents a dialog which permits you to: turn *netmon* polling on and off, configure the interval for *netmon* to delete nodes which are down, turn *netmon* discovery on and off, configure *netmon* for auto-adjusting or fixed polling intervals, and configure the *netmon* configuration polling interval. The polling intervals may only be modified by users with write permission for the directory **/usr/OV/databases/openview/topo**. For more information about this dialog, press the help button in this dialog. See also *netmon* (1m).

Note: functionality related to monitoring OpenView daemons has been moved to *xnmevents*(1).

Options

nmpolling takes no options.

AUTHOR

nmpolling was developed by HP.

FILES

/usr/OV/registration/\$LANG/nmpolling OpenView Windows application registration file

SEE ALSO

ovw(1), *netmon*(1m), *xnmevents*(1)

EXTERNAL INFLUENCES

International Code Set Support

Supports 8-bit and 16-bit data.

NAME

ovhelp - OpenView Windows help subsystem

SYNOPSIS

ovhelp

DESCRIPTION

ovhelp implements the OpenView Windows help subsystem responsible for displaying *ovw* help screens and application help screens via *OVwShowHelp(3)*. *ovhelp* is registered to run throughout each OVW session.

Help File Format

The help files which *ovhelp* are plain text files. The first line of a help file is used for the title line in the help screen. For example, if the file contained:

Automatic Layout

Purpose

=====

Each submap may have a layout algorithm that determines how symbols are placed on the submap. Automatic layout enables or disables the layout algorithm.

Then *Automatic Layout* is used for the help screen title and the rest of the text is presented in the help text window.

Help Index File Format

The syntax of the help index file is a filename, followed by a tab character, followed by text to use for the title in the index, i.e.:

FileName<tab>*Index Entry*

The *FileName* should be a full path name for an *ovhelp* help file.

There are two special cases for *FileName*. If set to **INDEXTITLE**, *ovhelp* assumes that the index entry text is the title of the index screen and should not appear in the index. If set to **TITLE**, *ovhelp* assumes that the index entry text is a title line within the index that should appear in the text window, but does not point to a help file.

An example help index can be found in */usr/OV/help/C/OVW/Functions/INDEX*. Here is an excerpt from that file:

```
*INDEXTITLE*<tab>OpenView Windows Function Index
*TITLE*<tab>Edit: Cut ->
/usr/OV/help/C/OVW/Functions/edcutall.fun<tab> From All Submaps
/usr/OV/help/C/OVW/Functions/edcutsub.fun<tab> From This Submap
```

The resulting index is titled *OpenView Windows Function Index* and the index screen contains the text:

```
Edit: Cut ->
  From All Submaps
  From This Submap
```

Resources

The following resources are used in the X-windows environment and may be specified using the standard X-windows resource specification methods:

PrintCommand

The command to use for the print button to print the help text. The default setting is */usr/bin/lp -s -onroff*.

SearchColor

The name of the color to use to highlight search results when *UseSearchColor* is True.

UseSearchColor

If **UseSearchColor** is **True**, then search matches are displayed in **SearchColor**. If **False**, the search results are not highlighted.

AUTHOR

ovhelp was developed by HP.

FILES

/usr/OV/registration/C/ovhelp	The OVW application registration file
/usr/OV/help/*	On-line help directories
/usr/lib/X11/app-defaults/OVHelp	Global X resources

SEE ALSO

ovw(1), *OVwShowHelp*(3).

WARNINGS

Only one instance of *ovhelp* should run as the OpenView Windows help subsystem. If additional instances of *ovhelp* are registered to be started by OpenView Windows as actions, each instance will ignore any command line arguments and register with OpenView Windows as the help application. OpenView Windows will send all help requests to each instance of *ovhelp*, resulting in multiple identical help windows for each help request.

EXTERNAL INFLUENCES**International Code Set Support**

Single- and multi-byte character code sets are supported for displayed help text. The search function only supports single-byte character code sets.

NAME

ovw - OpenView Windows graphical user interface

SYNOPSIS

ovw [-ro|-rw] [-map *map_name*]

ovw -config

ovw -verify

ovw -fields

DESCRIPTION

The *ovw* program is the OpenView Windows graphical user interface for the Openview network and systems management environment. Openview Windows is the graphical user interface into the Openview network and systems management environment. It provides a graphical map of the environment being managed. The map contains a hierarchy of submaps accessible via symbols and a set of pull-down menus for access to map operations and to applications.

For detailed information on all aspects of the map and the accompanying operations, consult the on-line Help system provided with *ovw*, available under the Help menu item on the menu bar.

ovw will only run after the OpenView Windows object database daemon *ovwdb*(1m) has first been started with *ovstart*(1m). See *ovwdb*(1m) for more information on the OpenView Windows object database.

At startup, OpenView Windows reads application and symbol registration files to define the symbols that are available for the map and the applications that are integrated into the menu bar and executable symbols. For more information on registration files, their syntax, and how they are processed, see *OVwRegIntro*(5).

For full details on using *ovw*, consult the *HP OpenView Windows User's Guide*.

Options

In addition to most standard X Toolkit command-line options, *ovw* supports the following options.

- config** compiles the bitmaps used for developer and user defined symbols. Compiling the bitmaps allows them to be read in quickly during startup of Openview Windows. When started with this option *ovw* will *not* start a normal "windowed" session. It will only process symbol registration files during this processing.
- fields** creates the fields defined in the currently installed field registration files in *ovw* object database. When started with this option *ovw* will *not* start a normal "windowed" session. It will only process field registration files during this processing.
- map *map_name*** Open map *map_name*. Create the map *map_name* if it does not exist. The default behavior is to open a configured default user map based on the user's UID. If no user default map is configured, *ovw* opens the map named *default*.
- ro** Open all maps as read-only. This option must not be specified at the same time as the **-rw** option.
- rw** Open all maps as read-write if possible. This option is the default behavior and is only useful if the X-Resources specify read-only as a default behavior.
- verify** Check the syntax for all the installed application, symbol type, and field registration files. Verify the semantics of the application registration files and report conflicts among application registration information. When started with this option *ovw* will *not* start a normal "windowed" session.

Resources

The following resources are used in the X-windows environment and can be specified using the standard X-windows resource specification methods:

mapBackground

This resource specifies the background color of the *ovw* submap windows.

logoName

This resource is an absolute path specification for the bitmap to placed in the lower right corner of each *ovw* submap window. The bitmap should be an X11 standard bitmap and should have be 120 pixels wide and 68 pixels high.

logoForeground

This resource specifies the foreground color to use in drawing the logos in the lower corners of *ovw* submap windows.

logoBackground

This resource specifies the background color to use in drawing the logos in the lower corners of *ovw* submap windows.

mapWidth

This resource specifies the initial default width of *ovw* submap windows.

mapHeight

This resource specifies the initial default height of *ovw* submap windows.

extraSpacing

This resource specifies the extra amount of space in pixels to put between the symbols on the following five dialog boxes: Legend, Add Object, LocateByStatus, LocateByType, and Change Symbol Type.

symbolInteriorColor

This resource specifies the color used to "fill" in the icon symbols used in *ovw*.

symbolEdgeColor

This resource specifies the color used to draw the basic internal patterns in the *ovw* icon symbols.

borderColor

This resource specifies the color used to draw the outer shapes of the *ovw* icon symbols.

labelColor

This resource specifies the color used in drawing the characters for the *ovw* symbols labels.

labelBackground

This resource specifies the background color used in drawing the labels for *ovw* symbols.

lineColor

This resource specifies the color used to draw miscellaneous lines such as the bus segment. It is not the color for a connection.

unmanagedStatusLineColor

This resource specifies the color used to draw connections with an unmanaged status.

testingStatusLineColor

This resource specifies the color used to draw connections with a testing status.

restrictedStatusLineColor

This resource specifies the color used to draw connections with a restricted status.

disabledStatusLineColor

This resource specifies the color used to draw connections with a disabled status.

unknownStatusLineColor

This resource specifies the color used to draw connections with an unknown status.

upStatusLineColor

This resource specifies the color used to draw connections with a normal (up) status.

warningStatusLineColor

This resource specifies the color used to draw connections with a warning status.

marginalStatusLineColor

This resource specifies the color used to draw connections with a marginal status.

majorStatusLineColor

This resource specifies the color used to draw connections with a major status.

downStatusLineColor

This resource specifies the color used to draw connections with a critical (down) status.

lineHighlightColor

This resource specifies the color used to draw a highlighted connection.

unmanagedStatusColor

This resource specifies the background color for icons with an unmanaged status.

testingStatusColor

This resource specifies the background color for icons with a testing status.

restrictedStatusColor

This resource specifies the background color for icons with a restricted status.

disabledStatusColor

This resource specifies the background color for icons with a disabled status.

unknownStatusColor

This resource specifies the background color for icons with an unknown status.

upStatusColor

This resource specifies the background color for icons with a normal (up) status.

warningStatusColor

This resource specifies the background color for icons with a warning status.

marginalStatusColor

This resource specifies the background color for icons with a marginal status.

majorStatusColor

This resource specifies the background color for icons with a major status.

downStatusColor

This resource specifies the background color for icons with a critical (down) status.

selectColor

This resource specifies the color to draw the "select" square around selected *ovw* symbols.

mapName

This resource specifies the map to open. This resource overrides an option specified using the *ovw* Configure Map menu. It is overridden with the *'-map'* option.

readOnly

This resource is used to specify how to open the map for user modifications. *True* indicates the map is to be opened read-only. *False* indicates the map is to be opened read-write (unless other factors dictate the map can only be opened read-only).

closeTimeout

If non-zero, this resource specifies how long (in seconds) *ovw* will wait for map applications to acknowledge the *ovwMapClose* event before *ovw* will continue without an application acknowledgement. If this resource is 0 (the default), *ovw* will wait forever. Closing the map without waiting for all map applications to acknowledge the *ovwMapClose* event is very likely to corrupt the map database.

queryAddSymbolTimeout

This resource specifies how long (in seconds) to wait for an application to respond to the *ovwQueryAddSymbol* event before *ovw* reports that the application did not respond.

queryConnectSymbolsTimeout

This resource specifies how long (in seconds) to wait for an application to respond to the *ovwQueryConnectSymbols* event before *ovw* reports that the application did not respond.

queryDeleteSymbolsTimeout

This resource specifies how long (in seconds) to wait for all queried applications to respond to the *ovwQueryDeleteSymbols* event before *ovw* continues to process the cut or delete operation. If the timeout expires then *ovw* determines which symbols it can cut or delete and which it must deny because of the application(s) that did not respond.

queryDescribeChangeTimeout

This resource specifies how long (in seconds) to wait for an application to respond to the *ovwQueryDescribeChange* event before *ovw* reports that the application did not respond.

queryAppConfigChangeTimeout

This resource specifies how long (in seconds) to wait for an application to respond to an *ovwQueryAppConfigChange* event before *ovw* reports that the application did not respond.

clickDelay

This resource specifies how close together (in milliseconds) successive mouse clicks must be to activate the double click behavior on the submap windows.

smallFont

This resource specifies the font to use for the smallest (many characters) size labels (should be an 8 point font).

size10Font

This resource specifies the font to use for small size labels (should be a 10 point font).

size20Font

This resource specifies the font to use for medium size labels (should be a 12 point font).

size30Font

This resource specifies the font to use for large (few characters) size labels (should be a 14 point font).

defaultRadius

This resource specifies the default radius for map symbols in pixels. This is the standard size used for scaling symbols. The maximum size of a symbol is derived from the default radius (the maximum radius is 5/3 of the default radius). The default is 15.

minimumSubmapBorder

This resource specifies the minimum border on the left and right sides of a submap window in pixels. If long symbol labels are used, this value can be increased to prevent labels from being obscured at the edge of the window. The default is 30.

submapDialog*XmlList*FontList

This resource specifies the fonts to use for the list of submaps in the "Submaps in Map" dialog box. The first value is the font to use for submaps which occur only once. Submaps which appear multiple times in the hierarchy use the initial font if this occurrence (in the list) is the one which will show the children of this submap. Only one of the multiple instances of a particular submap will appear in the "standard" font. The other instances will be shown with the font specified with the "=duplicate" specification. For more information on FontList specification see the X11 and Motif documentation.

dragInitiatorProtocolStyle

This resource is set to *DRAG_NONE* so that in *ovw* no widgets are enabled as initiators for (Motif 1.2) drag transfer.

dragReceiverProtocolStyle

This resource is set to *DRAG_NONE* so that in *ovw* no widgets are enabled as receivers for (Motif 1.2) drag transfer.

Object Database Fields

The following fields are defined in the OVW field registration file and are available for developer use. These fields are placed in the OVW object database when *ovw -fields* is run. For more information on *ovw* field registration see *OVwRegIntro(5)*.

OVW Registration Fields

Object Purpose	OVw Db Field Name	OVw FieldType	Field Flags
Global Capabilities	"vendor"	ovwEnumField	Capability General Locate
Global Name Fields	"Selection Name" "IP Hostname"	ovwStringField	Name Loc
Location Capabilities	"isNetwork" "isInternet"	ovwBooleanField	Capability
Segment Capabilities	"isSegment" "isBusSegment" "isStarSegment" "isTokenRingSegment" "isFDDIHubSegment" "isSerialSegment"	ovwBooleanField	Capability
Nodal Capabilities	"isNode" "isDevice" "isComputer" "isConnector" "isBridge" "isRouter" "isHub" "isRepeater"	ovwBooleanField	Capability
Computer Types	"isPC" "isWorkstation" "isMini" "isMainFrame"	ovwBooleanField	Capability
Device Types	"isPrinter" "isAnalyzer"	ovwBooleanField	Capability
Connection Capabilities	"isConnection"	ovwBooleanField	Capability
Interface Capabilities	"isCard" "isInterface" "isStarInterface" "isThinInterface" "isThickInterface" "isSerialInterface"	ovwBooleanField	Capability
Client/Server Capabilities	"isClient" "isServer"	ovwBooleanField	Capability
Software Capabilities	"isSoftware"	ovwBooleanField	Capability
IP Specific	"isIP"	ovwBooleanField	Capability

EXTERNAL INFLUENCES
Environment Variables

ovw sets the environmental variable LANG for itself and the other processes it starts by reading the file `/usr/ovw/conv/LANG` which, if present, should contain only the single-word value to which LANG should be set. LANG determines the language in which messages appear. If `/usr/ovw/conv/LANG` does not exist, and if LANG is not set or is set to the empty string, a default of "C" (see `lang(5)`) is used.

DISPLAY determines the host and display number to use when contacting an X server.

XENVIRONMENT determines the name of a resource file for overriding global *ovw* X resources.

International Code Set Support

Single- and multi-byte character code sets are supported.

AUTHOR

ovw was developed by HP.

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/help/*	On-line help directories
/usr/OV/registration/*	Application registration directories
/usr/OV/symbols/*	Symbol type registration directories
/usr/lib/X11/app-defaults/OVw	Global X resources

SEE ALSO

bitmap(1), xrbdb(1), ovstart(1m), ovwdb(1m), OVwRegIntro(5),
HP OpenView Windows User's Guide, X11R4 documentation, and OSF Motif documentation.

NAME

ovwperms - OpenView Windows map permission listing and changing

SYNOPSIS

```
ovwperms [-a] [-G] [-l] [-m mode] [-u user] [-g group] [map ...]
ovwchown [-a] [-G] user [map ...]
ovwchgrp [-a] [-G] group [map ...]
ovwchmod [-a] [-G] mode [map ...]
ovwls [-a] [-G] [map ...]
```

DESCRIPTION

The *ovwperms* command is used to view and modify permission associated with Openview Windows maps as created with *ovw(1)*. Permissions are maintained similar to normal file system permissions, except that permissions are forced to be the same across all files associated with a given *map*, and are based on user ID and group ID of the executing *ovw(1)* process. The *ovwchown*, *ovwchgrp*, *ovwchmod*, and *ovwls* commands are equivalent to using the *-u*, *-g*, *-m*, and *-l* options to *ovwperms*, respectively.

The change the permissions for a given map, you must be the owner of the map or the superuser. To see the current permissions use the *-l* option.

Options

The *ovwperms* command supports the following options:

- a** Apply the operation to all existing maps. Default if the *-l* option is used.
- g group** Change the group ID of the *maps* to *group*. The group can be either a decimal group ID or a group name found in the group file.
- G** Apply the operation to the global map permissions. The global map permissions control the ability to create new maps or delete existing ones.
- l** List the current permissions for the indicated *maps*.
- m mode** Change the permissions of the *maps* according to *mode*, which can be an absolute or symbolic value as accepted by *chmod(1)*.
- u user** Changes the owner of the *maps* to *user*. The user can be either a decimal user ID or a login name found in the password file.

At least one of *-g*, *-l*, *-m*, or *-u* must be specified or implied by the invocation name of the command. If the *-l* option is not used, at least one of *-a*, *-G*, or a *map* must be specified.

EXAMPLES

The following command changes the owner of all maps to *ovwuser*, the group of all maps to *ovwgroup*, and sets permissions of all maps to remove write permission from others. The example assumes that a single user owns all maps, or that the command is run by the super-user.

```
ovwperms -a -u ovwuser -g ovwgroup
```

The following commands are equivalent, and change the owner of maps *MyMap1* and *MyMap2* to *ovwuser2*.

```
ovwperms -u ovwuser2 MyMap1 MyMap2
ovwchown ovwuser2 MyMap1 MyMap2
```

EXTERNAL INFLUENCES

International Code Set Support

Supports 8-bit and 16-bit data, 8-bit filenames.

AUTHOR

The *ovwperms* command was developed by HP.

SEE ALSO

ovw(1), *chmod(1)*, *chgrp(1)*, *chown(1)*, *ls(1)*, *passwd(4)*, *group(4)*

NAME

rbdf - check remote disk space using SNMP requests

SYNOPSIS

rbdf [-i] [-c *community*] [*node*]

DESCRIPTION

rbdf lists the amount of free disk space for all file systems locally mounted on *node*. The output format is similar to *bdf*(1M), except *rbdf* does not list disk space for NFS mounted systems. The numbers are reported in kilobytes.

The *node* must be a remote node supporting SNMP requests; the default *node* is the local node.

The graphical interface equivalent to *rbdf* is the Request Disk operation; execute the graphical interface using *xnm*(1).

Options

-i Report the number of used and free inodes.

-c *community*

Override the SNMP community name (as configured in */usr/OV/conf/ovsnmp.conf*) to *community*. See *ovsnmp.conf*(4).

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *rbdf* behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

DEPENDENCIES**SNMP and HP-UX**

rbdf uses HP-UX-specific SNMP objects and only works with HP-UX nodes.

AUTHOR

rbdf was developed by HP.

FILES

/usr/OV/conf/ovsnmp.conf
SNMP configuration

SEE ALSO

ovw(1), *bdf*(1M), *snmpd*(1M), *ovsnmp.conf*(4).

*/usr/OV/doc/** (for related RFC's)

NAME

rnetstat - list network information of a remote SNMP node

SYNOPSIS

rnetstat [-a|-A|-e|-i|-I|-r|-S|-u] [-xlnNd] [-c *community*] [-t *timeout*] [-R *retries*] [*node*]

DESCRIPTION

rnetstat symbolically lists the contents of various network-related data structures on the target *node*. The information appears in various formats, depending on the options specified.

By default, the TCP 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 symbolically appear according to the databases *hosts*(4) and *networks*(4). If a symbolic name for an address is unknown, or if the -n option is specified, the address is printed in Internet dot format (see *inet*(3N)). Unspecified and wild-card addresses and ports appear as an asterisk (*).

The graphical interface equivalent to *rnetstat* is the Request TCP Connections operation; execute the graphical interface using *xnm*(1).

Options

- a Show all TCP and UDP table entries; generally, sockets used by server processes are not shown.
- A Show the ARP cache entries; these entries include the IP address to physical (link level) address mappings.
- e Show ethernet (physical or link level) statistics, including interface name, number of frames received and transmitted, number of input CRC 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 similar to that provided by *landiag*(1M).
- i Show the network interfaces and their associated attributes and statistics; includes interface name, input packets, input errors, output packets, and output errors.
- I Show 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 the default table format. When the -l option is specified, more information is provided if available. Applies only to the -e option.
- n Show network addresses as numbers rather than listing them symbolically.
- N Do not truncate names. By default, names are truncated to align the output columns.
- r Show 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 Show 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 appears.
- u List the UDP table (local address and port information).
- c *community*
Override the SNMP community name (as configured in */usr/OV/conf/ovsnmp.conf*) to *community*. See *ovsnmp.conf*(4).
- t *timeout*
Override the default SNMP timeout, in tenths of a second.
- R *retries*
Override the number SNMP retries.

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty

string, a default of "C" (see *lang(5)*) is used instead of LANG. If any internationalization variable contains an invalid setting, *rnetstat* behaves as if all internationalization variables are set to "C." See *environ(5)*.

International Code Set Support

Supports single-byte character code sets.

DEPENDENCIES**SNMP and HP OpenView TCP/IP Agent**

The target *node* must support SNMP. Most operations use only SNMP objects as defined in RFC 1156. The **-a** and the **-u** options use SNMP objects as defined in RFC 1158. The **-e** option uses HP OpenView TCP/IP Agent-specific SNMP objects.

AUTHOR

rnetstat was developed by HP.

FILES

/usr/OV/conf/ovsnmp.conf_db/*
SNMP configuration

SEE ALSO

netstat(1), *landiag(1M)*, *snmpd(1M)*, *ovsnmp.conf(4)*, *hosts(4)*.
/usr/OV/doc/* (for related RFC's)

NAME

rping - Send remote packet echo using ICMP Echo requests

SYNOPSIS

rping [-d] [-c *community*] [-n *count*] [-s *size*] [-t *timeout*] [*source*] *target*

DESCRIPTION

rping is similar to *ping*(1M), except that it allows the ICMP Echo request to be initiated from a remote node. An SNMP request is sent once per second to the *source* node requesting that the *source* node send one ICMP Echo packet to the *target* node, and return the round-trip time for the ICMP request. For each response received, *rping* outputs the round-trip time, in milliseconds, for the response to reach the *source* node. The *rping* command terminates when a SIGINT signal is received, typically as a result of pressing the Break key. When *rping* terminates, it reports statistics concerning the number of packets sent and lost, and a summary of the round-trip times.

Specify the *source* and *target* nodes as either host names or Internet addresses. All specified symbolic names are looked up in the host name database (see *hosts*(4)). Internet addresses must be in the standard Internet dot notation (see *inet_addr*(3N)). The default *source* is the local node.

Options

- c *community* Set the SNMP community name to *community*; overrides the community name in */usr/OV/conf/ovsnmp.conf*. See *ovsnmp.conf*(4).
- n *count* Set the number of packets for *rping* to transmit before terminating. The range is 1 to (2**31 -1), decimal. By default *rping* continues sending packets until interrupted.
- s *size* Set the size of ICMP packets transmitted to *size*. The default size of transmitted packets is 64 bytes. The minimum value allowed for *size* is 8 bytes, and the maximum is 2012 bytes. If *size* is less than 16 bytes, there is not enough room for timing information, and the round-trip times do not print.
- t *timeout* Specify how long, in seconds, the *source* node waits for the ICMP response. The *source* node does not service other SNMP requests during this time. The range is 5 to 60 seconds. The default is 5 seconds.

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *rping* behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

DEPENDENCIES**SNMP and HP OpenView TCP/IP Agent**

rping uses HP OpenView TCP/IP Agent-specific SNMP objects. If the *source* node does not support these objects, an error occurs.

AUTHOR

rping was developed by HP.

FILES

/usr/OV/conf/ovsnmp.conf
SNMP community names

SEE ALSO

ping(1M), *snmpd*(1M), *ovsnmp.conf*(4)

NAME

snmpColDump - ASCII dump/modify data collected by *snmpCollect*(1M)

SYNOPSIS

snmpColDump [-tTipfsoiu] [-I Object-ID] [-L Object-ID] filename **snmpColDump** -r asciiFileName

DESCRIPTION

snmpColDump formats binary data files in */usr/OV/databases/snmpCollect* that were previously created by *snmpCollect*(1M) into human-readable and machine-parseable ASCII format. It also allows supports editing of the binary data.

For each MIB variable being collected, *snmpCollect* creates two files in */usr/OV/databases/snmpCollect*:

- A *binary data* file containing the data collected for that variable. The binary file has the same name as the MIB label specified in the "MIB Data Collection" menu. It is the path to the binary data file that is given as an argument to *snmpColDump*. If instance wildcarding is in effect, an instance number will be appended to the filename.
- A separate *ID* file containing the units, syntax, and object ID for the MIB variable. This ID file has the same name as the binary data file, except there is a ! appended to the filename. The information in the ID file is also available via various options to *snmpColDump*.

The default output from *snmpColDump* consists of records, one per line, containing three fields:

- Date and time when the data was collected,
- Domain name of the node on which the data was collected,
- Value of the collected MIB variable. MIB variables of type COUNTER are printed in floating-point notation. INTEGER and GAUGE types are printed as whole (non floating-point) numbers.

Options

The following options are used to add additional fields to *snmpColDump* output. To remove an unwanted field, use *awk*(1).

- t Additionally display the time the data was collected as seconds since 1970. This makes sorting records easier.
- T Additionally display the start time as seconds since 1970.
- I Additionally display the official IP address of the node.
- p Additionally display the collection period in seconds.
- I Additionally display an instanceString which is obtained via SNMP. SNMP getNext requests are made to the current node using this object ID and the instance of the Object-ID whose value matches the instance of the collection is displayed. See examples below for use.
- L Additionally display a value which is obtained via SNMP. The instance is appended to the Object ID given as the argument to this option. An SNMP get request is made to the current node using this object ID and the resulting value is displayed. Note that embedded blanks and tabs may be in this returned value, which is why it is the last field. See examples below for use.
- r Replace the binary file *filename* with the binary equivalent of *asciiFile*. If *asciiFile* is "-", standard input is used. This allows truncation of files, or modification of the contents of the file. The *asciiFile* must be an ASCII file in the form of lines containing
 <startTime> <endTime> <Dotted IPaddress> <value>
 where <startTime> and <endTime> are integers representing time since the epoch (time_t format), <Dotted IPaddress> is the string containing the IP address, and <value> is the double precision value for the interval. An example of this form can be obtained using the command
 snmpColDump -tTI FILE |
 awk -F\t '{printf("%d\t%d\t%s\t%lg\n", \$4, \$5, \$6, \$3)}'
 This allows deleting of old entries in the binary data base file.

The following options to *snmpColDump* help you gather additional, specific information for generating reports, but produce no data:

- s Print type (syntax) string of the MIB variable.

- o Display objectID string of the MIB variable.
- i Display instance of the MIB variable.
- u Display units string of the MIB variable.

The following options to *snmpColDump* allows you to continuously monitor the data collected by *snmpCollect(1m)*:

- f Follow the file, sleeping until more data is available to display.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang(5)*) is used instead of LANG. If any internationalization variable contains an invalid setting, *snmpColDump* behaves as if all internationalization variables are set to "C." See *environ(5)*.

International Code Set Support

Supports single-byte character code sets.

EXAMPLES

The following examples assume that data have been collected using *snmpCollect(1M)* under the label **macDeferred.1**. It also assumes use of *sh(1)* or *ksh(1)*.

To print out a the average value for node Moe:

```
snmpColDump /usr/OV/databases/snmpCollect/macDeferred.1 |
awk -F\t '/Moe/{num++; sum+=$3} END{print sum/num}'
```

To retrieve information about node "Curly":

```
snmpColDump /usr/OV/databases/snmpCollect/macDeferred.1 | fgrep Curly
```

To additionally display the description of the collected interface:

```
snmpColDump -L interfaces.ifTable.ifEntry.ifDescr /usr/OV/databases/snmpCollect/macDeferred.1
```

To additionally display the IP address of the interface:

```
snmpColDump -l ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex
/usr/OV/databases/snmpCollect/macDeferred.1
```

To keep only the last 2000 entries in file 1MinLoadAvg (you must be root to run *snmpColDump -r*):

```
lineno='snmpColDump 1MinLoadAvg | wc -l'
if [ $lineno -gt 2000 ]; then
    lineno='expr $lineno - 2001'
else
    lineno=1
fi
snmpColDump -tTI 1MinLoadAvg | sed -n $lineno,$p' |
awk -F\t '{printf("%d\t%d\t%s\t%lg\n", $4, $5, $6, $3)}' |
snmpColDump -r - 1MinLoadAvg
```

To keep only data from nodes Moe, Larry, and Curley (substitute the TAB character for <TAB>, each name must be on a separate line):

```
snmpColDump -tTI 1MinLoadAvg | grep '\
<TAB>Moe<TAB>
<TAB>Larry<TAB>
<TAB>Curley<TAB>' |
awk -F\t '{printf("%d\t%d\t%s\t%lg\n", $4, $5, $6, $3)}' |
snmpColDump -r - 1MinLoadAvg
```

To delete node Moe from the database (substitute the TAB character for <TAB>):

```
snmpColDump -tTI 1MinLoadAvg | grep -v '<TAB>Moe<TAB>' |
awk -F\t '{printf("%d\t%d\t%s\t%lg\n", $4, $5, $6, $3)}' |
snmpColDump -r - 1MinLoadAvg
```

To print the nodename, number of points, and average value of file ifInOctets.1:

```
snmpColDump ifInOctets.1 | awk 'BEGIN{numNodes=0} {
    for (i=0; i<numNodes; i++)
        if (nodes[i] == $3)
            break;
    if (i == numNodes) {
        total[numNodes] = numPoints[numNodes] = 0
        nodes[numNodes++] = $3
    }
    total[i] += $4
    numPoints[i]++
}
END{for (i=0; i<numNodes; i++)
    printf("%s\t%d\t%g\n", \
        nodes[i], numPoints[i], total[i]/numPoints[i])
}'
```

To import the data from an ifInOctets.1 file into a spreadsheet (such as Lotus) which requires comma separated fields, use the following (substitute the TAB character for <TAB>):

```
cd /usr/OV/databases/snmpCollect
snmpColDump ifInOctets.1 | sed 's/<TAB>/,/g'
```

Note that if you need a numerical value for time instead of a string value, use the -t option.

To import the data about node "Curly" from an ifInOctets.1 file into a spreadsheet (such as Lotus) which requires comma separated fields, use the following (substitute the TAB character for <TAB>):

```
cd /usr/OV/databases/snmpCollect
snmpColDump ifInOctets.1 | sed 's/<TAB>/,/g' | grep 'Curly,'
```

WARNINGS

When using the -l or -L options, if the SNMP request cannot get the requested data it will make up a string. This string is (*Instance_n*) for the -l option or *instance_n* for the -L option.

AUTHOR

snmpColDump was developed by HP.

FILES

```
/usr/OV/databases/snmpCollect/*
/usr/OV/databases/snmpCollect/*[!]
```

SEE ALSO

snmpCollect(1M), awk(1), sort(1), mib.coerce(4).

NAME

snmptrap - issue an SNMP Trap

SYNOPSIS

snmptrap [*options*] *node* *enterprise* *agent-addr* *generic-trap* *specific-trap* *time-stamp* [*variable type value* ...]

DESCRIPTION

snmptrap issues an SNMP Trap to *node*. *node* can be either an Internet address or a host name (see *hosts*(4)).

enterprise has the format of .A.B.C.D..., where A, B, C, and D are subidentifiers in decimal notation. An *enterprise* of "" (empty string) is interpreted as the default object identifier as appropriate, based on the hardware in use.

agent-addr must be either an Internet address or a host name (see *hosts*(4)). An *agent-addr* of "" (empty string) is interpreted as the local hostname.

generic-trap must be a whole number between zero and six (inclusive).

specific-trap can be any 32-bit integer.

time-stamp 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.

Each *type* must be one of the following types: integer octetstring octetstringhex octetstringoctal octetstringascii objectidentifier null ipAddress counter gauge timeticks opaque opaquehex opaqueoctal opaque-ascii.

See RFC 1155 for a complete description of each *type*.

The *value* must be valid for the *type* specified. 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. *value* must occupy not more than 512 bytes maximum.

Options

- d Dump (list) input and output packets in a hexadecimal and partially ASN.1 decoded format.
- p Override the default port for sending/receiving to *port*.
- c Override the SNMP community name (as configured in */usr/OV/conf/ovsnmp.conf*) to *community*. See *ovsnmp.conf*(4).

EXTERNAL INFLUENCES**Environment Variables**

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *snmptrap* behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

EXAMPLES

The following command sends an SNMP trap to node *testnode* setting the agent and enterprise to the machine upon which the command is run and with *time-stamp* equal to 15. The trap has *generic-trap* equal to 6, and *specific-trap* equal to 15. The information passed with the trap is *system.sysDescr.0*.

```
snmptrap testnode "" "" 6 1 15 system.sysDescr.0 \
    octetstringascii "SunOS hpcndnw2 4.1 1 sun4c"
```

AUTHOR

snmptrap was developed by Carnegie-Mellon University and HP.

SEE ALSO

snmpd(1M), *trapd*(1M), *trapd.conf*(4)
*/usr/OV/doc/** (for related RFC's)

NAME

snmpwalk - query a node repeatedly using SNMP GetNext requests
 snmpnext - query a node using SNMP GetNext request
 snmpget - query a node using SNMP Get request
 snmpset - issue an SNMP Set request

SYNOPSIS

snmpwalk [*options*] *node* [*variable*]
snmpnext [*options*] *node* *variable* [...]
snmpget [*options*] *node* *variable* [...]
snmpset [*options*] *node* *variable* *type* *value* [...]

options: [-d] [-ttimeout] [-rretries] [-pport] [-ccommunity]

DESCRIPTION

snmpwalk uses the SNMP GetNext request to query *node* for information. The value of *variable* determines the portion of the object identifier 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. *snmpwalk* terminates when the variable returned is no longer in the subtree indicated by *variable*. *snmpnext* performs the same action as *snmpwalk*, except only returns one value. The default *variable* for *snmpwalk* is *.iso.org*.

snmpget uses the SNMP Get request to query *node* for information.

snmpset issues an SNMP Set request to alter MIB objects on the remote *node*.

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" appears.

Options

The default values for the options are determined by the SNMP Configuration database */usr/OV/conf/ovsnmp.conf_db*. See *snmpconf(1)* and *ovsnmp.conf(4)*.

- d Dump to standard output all SNMP packets in a hexadecimal and decoded ASN.1 format.
- t Override the default timeout with *timeout*, given (in 1/10th of a second units).
- r Override the default number of retries to *retries*, with the given value.
- p Override the default port where requests are sent to *port*.
- c Override the SNMP community name for the target *node*.

An exponential backoff algorithm is used based on *timeout* and *retries* to resend the SNMP request if a response is not received in the *timeout* period of time. For example, if *timeout* is 10 (1 second) and *retries* is 3, the initial timeout would occur at 1 second, the first retry would timeout after 2 seconds, the second retry would timeout after 4 seconds, and the last retry would timeout after 8 seconds. In this case, the total elapsed time before the command terminates with a no response message would be 15 seconds.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages appear. If LANG is not specified or is set to the empty string, a default of "C" (see *lang(5)*) is used instead of LANG. If any internationalization variable contains an invalid setting, the command behaves as if all internationalization variables are set to "C." See *environ(5)*.

International Code Set Support

Supports single-byte character code sets.

EXAMPLES

The following command requests the system subtree for node *testnode*. **snmpwalk testnode system**

Output for the above command typically resembles the following:

```
system.sysDescr.0 : DISPLAY STRING- (ascii): SunOS testnode
system.sysUpTime.0 : Timeticks: (1123049528) 129 days, 23:34:55.28
```

```
system.sysName.0 : DISPLAY STRING- (ascii): testnode
system.sysLocation.0 : DISPLAY STRING- (ascii): Main Building, SE
system.sysServices.0 : INTEGER: 72
```

The following command sets the system contact to **Bob Jones** for the node *testnode*.

```
snmpset testnode system.sysContact.0 octetstring "Bob Jones"
```

The output for the above command is as follows.

```
system.sysContact.0 OCTET STRING-(ascii): Bob Jones
```

AUTHOR

snmpwalk, *snmpnext*, *snmpget*, and *snmpset* were developed by Carnegie-Mellon University and HP.

SEE ALSO

snmpd(1M), *xnmsnmpconf*(1), *ovsnmp.conf*(4)

*/usr/OV/doc/** (for related RFC's) i

NAME

xnmappmon - OpenView Terminal Output Encapsulation Tool

SYNOPSIS

xnmappmon [options] -cmd *shell_command*

DESCRIPTION

xnmappmon **xnmappmon** is the standard OpenView Graphic User Interface for encapsulating output of terminal-based commands in a consistent, integrated, single look and feel presentation. Together with the OpenView Windows menu registration, *xnmappmon* enables existing management applications to be integrated into OpenView Windows quickly and easily.

The *xnmappmon* application dialog consists of the following presentation areas

Title The dialog title may be specified as an *xnmappmon* command line option, or may be extracted from the application output.

Menu Bar

The menu bar contains menu items to print, save and sort the application output. On-line help for the application is also accessible from the menu bar.

Selection Name

This field is displayed when the -appendSelectList option is used. It contains the selection name or names passed to the application command (ie, target node name).

Heading

This field is useful to label the results displayed from the encapsulated command; especially if the results are in tabular form. The results heading may be specified as an *xnmappmon* command line option, may be extracted from the application output, or left blank.

Results

This area of the application dialog is used to display output from the encapsulated command.

Messages

This area of the application dialog is used to display error messages from the encapsulated command.

OPTIONS

xnmappmon supports the following options:

-cmd *shell_command*

This is the only required option. This must also be the last option specified. All command line arguments following this option are treated as arguments to *shell_command*. This is the command executed by *xnmappmon* whose output is displayed in the application dialog window.

-helpFile *filename*

The name of a file which resides relative to the directory */usr/OV/help/<LANG>/*. This file is used by the OpenView help system when the "Help→On Application" menu item in the application dialog is selected.

-node *nodename_list*

Same as -selectList. Provided for backward compatibility only.

-selectList *selection_list*

Use *selection_list* as the argument(s) appended to *shell_command* when -appendSelectList is specified. The *selection_list* entries may be separated by a comma, space, tab, or newline. However, *selection_list* must be enclosed in quotes (or double quotes) if any separator other than a comma is used. Only use this option when executing *xnmappmon* from the shell command line. The *selection_list* is derived automatically from the OpenView Windows map when *xnmappmon* is executed from the OpenView Windows menu bar.

-appendSelectList

Append entries from *selection_list* to *shell_command* according to the behavior of -singleDialog or -multipleDialogs.

-appendSelectListToTitle

Append the selection list entry for *shell_command* to the dialog title. When the -singleDialog option is specified, only the first entry in the selection list is appended to the dialog title. When the -multipleDialogs option is specified, each entry in the selection list is appended to the respective dialog title.

-multipleDialogs

When used with the `-appendSelectList` option, each entry in the *selection_list* is passed to a separate instance of the *shell_command*, with the result of each command displayed in a separate application dialog. This is the default behavior when `-appendSelectList` is specified. If `-appendSelectList` is not specified, this option is ignored.

-singleDialog

When used with the `-appendSelectList` option, the entire selection list is appended to *shell_command* and the result is displayed in a single application dialog. If `-appendSelectList` is not specified, this option is ignored.

-commandTitle title

Title to use for the application dialog.

-titleLine line_number

Which line of command output to extract as the application dialog title. This option supercedes `-commandTitle`.

-commandHeading heading

Heading to use for the application results.

-headingLine line_number

Which line of command output to extract as the heading. This option supercedes `-commandHeading`.

-dataLine line_number

Which line of command output to begin using as application results. A value of 3 for *line_number* will cause the first 2 lines to not be displayed.

-maxLines lines

Change the maximum number of data lines displayed. If *lines* is greater than zero, only the last *lines* number of lines are displayed. By default, all command output lines beginning with the `-dataLine` line are displayed.

-followOutput

Continuously display the end of the application results as new output is encapsulated. The default behavior is to display only the first portion of results and allow the user to move the scroll bars to view the additional results.

-tabWidth width

Expand embedded tabs in the command output to *width* spaces. The default tab width is 8.

-unbufIO

Display application results using unbuffered I/O. By default, buffered I/O is used for performance. In some cases, however, the application results may not be displayed immediately. Applications which generate output over an extended period of time, generate output in bursty patterns, or pause between generating output are most likely to require this option.

-clearOutput string

When the command output line exactly matches *string*, *xnmappmon* clears the application results instead of displaying the line. This option is useful when encapsulating the output of a command which iterates over some period of time, periodically refreshing the results with new ones.

Note, the `-unbufIO` option is most likely needed with this option. Also, *shell_command* must use unbuffered I/O or `fflush(3)` when printing results to `stdout` and `stderr`. Otherwise, command output may not be displayed in the application results immediately.

-clearError string

When the command messages (`stderr`) line exactly matches *string*, *xnmappmon* clears the application messages instead of displaying the line.

-clearDataLine line_number

When used with `-clearOutput`, resume displaying application results beginning with the *Nth* command output line following the clear indication. The default is 1 (ie, the line immediately following the clear indication). Use 0 (zero) to imply the same line as specified in the `-dataLine` option.

-stopSignal signal

Terminate *shell_command* with *signal* when the Stop button is pressed. The *signal* must be the numeric value of the signal; do not use the mnemonic signal name. By default, *xnmappmon* uses signal 2

(SIGINT) for this purpose.

-lingerOnStop

Allow command output to linger when the *Stop* button is pressed to terminate the *shell_command*. Use this option if the command displays concluding information when the command is terminated (ie, *ping(1)* displays summary statistics).

The following options control actions which may be performed on the *shell_command* output.

-print

Print results after command is executed.

-printerCommand command

Command to which the data output is piped.

-save

Save results after command is executed.

-saveFile filename

The name of a ASCII file to which the screen contents are dumped when using "File→Save" menu item is selected.

-sort Display the command output in sorted order

-reverseSort

When used with the **-sort** option, the sorted command output is displayed in reverse order. If **-sort** is not specified, this option is ignored.

-sortColumn1 column1

The beginning column on which to sort the command output.

-sortColumn2 column2

The ending column on which to sort the command output.

-exit Exit *xnmappmon* after *shell_command* is executed, and after any **-sort**, **-print**, or **-save** options are 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.

-timeOut seconds

This option is no longer necessary and is, therefore, ignored.

RESOURCES

Resources are documented in the */usr/OV/app-defaults/XNm* resource file.

EXAMPLE

To encapsulate the output of *ovstatus(1M)* use the command

```
xnmappmon
    -commandTitle "Status of OpenView Processes" -commandHeading "Status" -cmd
    /usr/OV/bin/ovstatus
```

EXAMPLE

Given a command which periodically polls selected devices and generates the following command output

```
Title Line
Heading Line
Data Line 1
Data Line 2
Data Line N
```

<pause between polls>

CLEAR

encapsulate the output so that the results are cleared and refreshed after each poll.

```
xnmappmon
    -appendSelectList -titleLine 1 -headingLine 2 -unbufIO -clearOutput CLEAR -clearData-
    Line 0 -cmd pollDevice
```

For other *xnmappmon* examples, refer to the /usr/OV/registration/C/ovip/NNM-IP.tbl OpenView Windows registration file which is part of the OpenView Network Node Manager product.

AUTHOR

xnmappmon was developed by HP.

FILES

/usr/OV/app-defaults/XNm

SEE ALSO

ovw(1), OVwRegIntro(5)

NAME

xnmevents - Load and Delete SNMP trap definitions

SYNOPSIS

xnmevents [-load *file*] [-event]

DESCRIPTION

xnmevents is the Motif user interface for browsing events. By giving it appropriate command line options, *xnmevents* can be instructed to load a trap definition file. *xnmevents* modifies the ASCII file */usr/OV/conf/\$LANG/trapd.conf*, which is a collection of trap definitions. See *trapd.conf(4)*. *xnmevents* has both a command-line mode, which is especially well suited for installation scripts, and an interactive mode, which can either run stand-alone as an X11/Motif application or is invoked when HP OpenView Windows (see *ovw(1)*) is started.

xnmevents also periodically checks to verify that certain HP OpenView daemons are running. See */usr/OV/app-defaults/XNm* for configuration information.

Using the *xnmtrap* user interface (available under Options→Event Configuration:SNMP), and the *xnmtrap.trapdConfFile* resource, *xnmtrap* can be used to create an importable file. Either start with an empty file pointed to by this resource, or a copy of the */usr/OV/conf/\$LANG/trapd.conf* file, and use the interface to create the importable file. That file can then be loaded at customization time using the *-load* option below.

Options

xnmevents supports the following options:

-load *FILE*

Merge *FILE* with */usr/OV/conf/\$LANG/trapd.conf*. Any definition which was previously defined will be replaced with the definition in *FILE*, unless the definition was defaulted in *FILE* and not defaulted in */usr/OV/conf/\$LANG/trapd.conf*. I.e., if the event already exists, and the new event has no description, then the old description will be used. If the field cannot be defaulted (such as severity or category), the old value will be used. If you need to replace these fields, delete the old event first, and all the new values will be used, since there will not be a duplicate event. Duplicate events are events with the same name, enterprise, generic trap, specific trap, and list of nodes.

If the event already exists under another name (i.e., same enterprise, generic, specific, and nodes, but different name), an error message will be generated. Use the *-delete* option to delete the old event.

-delete *NAME*

The action whose title is *NAME* is deleted, if it exists. Otherwise, the trap *NAME* is deleted, if it exists.

-event

Send the event that the *trapd.conf* file has been changed. This will cause *xnmevents* and *trapd(1m)* to reload */usr/OV/conf/\$LANG/trapd.conf*.

AUTHOR

xnmevents was developed by HP.

FILES

/usr/OV/conf/\$LANG/trapd.conf
/usr/OV/app-defaults/XNm

SEE ALSO

ovw(1), *trapd(1m)*, *trapd.conf(4)*.

NAME

xnmgraph - OpenView Line Graphing Tool

SYNOPSIS

xnmgraph [<options>] [node ...]

DESCRIPTION

xnmgraph is a Motif based graphing utility which can graph data collected by *snmpCollect(1m)*, live SNMP data, or user supplied data. *xnmgraph* allows the user to graph real-time and historical data, and to control many aspects of how the data is viewed.

The following options are available. See */usr/OV/app-defaults/XNm* for description of corresponding resources. The only required option is one of either **-mib** or **-mibFile**.

-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 colon separated list of options, in the following form, all one argument. The general form of the specification is:

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

mibOID:

MIB Object in dotted notation (without instance). If the form of this object is a '-', '+', '%', or '=' followed by a number, then standard input to *xnmgraph* is used as a stream of data. See DATA INPUT EXTENSIBILITY. MibOID can be '*' if the **-browse** option is used, indicating to get all MIBs from database and perform no SNMP. See *snmpCollect(1m)*. To graph a MIB expression, this field should be the name of a previously defined MIB expression. See *mibExpr.conf(4)*.

miblabel:

Label to give the object (defaults to the textual last component of mibOID or line number when using data input extensibility).

instanceRE:

Regular expression of matching instances. This defaults to ".*", which means to match any and all instances detected via the SNMP GETNEXT command. *xnmgraph* builds a menu of all possible instances by performing a SNMP GETNEXT on the MIB variable to find all the instances for a given node. Uses extended regular expressions. See *regexp(5)*. Caution: the '.' character is a regular expression. If it really is the character desired, escape it using a backslash ("\."). The instance 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 to be an exact instance which will be used. No SNMP GETNEXT will be performed.

instMatchOID:

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*. See *instMatchRE* below.

instMatchRE:

Regular expression matched against the results of a 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) which determines if a particular object should be graphed. As an example, when graphing *ifInOctets*, only certain types of interfaces make sense to graph. 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". 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 "[^1]\$|^1.\$|^2[^4]\$|^3-9.\$". This will find any single digit number but 1, or any 2 digit number starting with 1, or and 2 digit number whose second digit is not 4, or any 2 digit number starting with 3 through 9. If the value returned is an enumerated value, the enum is converted to a number, not the enumerated string. Uses extended regular expressions. See *regexp(5)*.

instLabelOID:

SNMP OID (minus instance) to query to get the label for the instance. The returned value is converted to a string and used as the label for the corresponding instance of this MIB variable. If the SNMPGET of this OID fails, or the OID is not supplied, the default instance label is the instance portion of the full OID. As an example, to get use ifDescr as the instance label, set *instLabelOID* to ".1.3.6.1.2.1.2.2.1.3".

instLabelTrunc:

Regular expression string 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" are not displayed. Uses extended regular expressions. See *regexp(5)*. As an example, if using *instLabelOID* to use the ifDescr as the instance label, the instance label may be "lan0 Hewlett-Packard DIO LAN Interface Hw Rev 98643". To remove the extraneous information, set *instLabelTrunc* to " ". This will instead use "lan0" as the instance label.

mult:

Multiplier of sampled value. Each sampled value is multiplied by this value (if present) before being displayed, allowing scaling of the data. This will be the default value which the user sees in the View→Line Configuration menu.

node:

Space separated list of nodes. If not present this defaults to node list passed at end of command line. If using DATA INPUT EXTENSIBILITY, this field can be used as a data name to cascade similar data sets together. If node is "*" and -browse option is used, the list of nodes is retrieved from the corresponding MIB variable in the database and perform no SNMP. See *snmpCollect(1m)*.

Multiple colon separated arguments are allowed, with commas separating each argument. ':' and ',' are not allowed in the colon separated fields.

-actualStats +actualStats

If a multiplier is given, don't show (show) statistics multiplied in the statistics dialog. Normally xnmgraph will present the statistical data as multiplied values, which can be modified via an option menu in the statistics dialog. This will change the initial state of that option menu.

-browse +browse

Turn on (off) browsing of collected data. When in the browse mode, no live SNMP data is performed, only data in the *snmpCollect(1M)* database is displayed. The "File→Update Data" menu item then becomes available.

-community community-name

Override the default community name defined by the *ovsnmp* library using the *ovsnmp.conf* file. Specifying this option will use the defined community name for all SNMP queries.

-c community-name

Same as -community.

-displayWidth <time>

Set the display width to <time>. Can use the characters s,m,h,d,w,y to indicate the units of <time>. The default is to display all of the data. If this option is used, the entry will also appear in the popup menu controlled by the right mouse button.

-dpyValList <list>

Comma separated list of initial displayed values in form line:value. Value is a bit-wise OR of the masks:

Minimum:1 Average:2 Maximum:4

indicating which values to initially display. Must be set to the range 1-7. As an example, *-dpyValList 1:3,2:4* will make the first line show Min&Avg, and the second line will show Max. The line numbers begin at 1.

-dump +dump

Turn on (off) dumping of SNMP packets to standard out.

-globalScale +globalScale

Turn on (off) scaling of y-axis based on the entire data set. If turned off, the scaling is done based only

on the displayed data. Turning this on is useful if the user will be paging through the data, so that the y-axis does not change from screen to screen.

-helpFile <help file>

The name of a file which resides relative to the directory `/usr/OV/help/<LANG>/`. This file will be used by the OpenView help system when the "Help→On Application" menu item is selected.

-hideMultiplier +hideMultiplier

If a multiplier is given, hide (don't hide) it in the legend. Normally **xnmgraph** will present the multiplier in the legend to indicate the actual values are not being graphed.

-lineWidth <pixels>

Set the initial line width for all lines to be `<pixels>` width. This number will not show up as a line width menu item unless the corresponding line width choices resource `xnmgraph.mutlValues` also includes this value.

-maxLines <num lines>

Specifies the maximum number of lines which will be displayed at one time. Note that this may be interactively incremented using the View→Add Line menu item.

-mibFile <file>

Instead of using the long `<line specification>` argument to `-mib`, the given file can contain the arguments. The contents of the file use the same format and type as the `-mib` option, except they are in a file. The file uses '#' as a comment character, and so the '#' character cannot be used in the colon separated fields. Leading and trailing white space of each line is removed, and each non comment line is concatenated together, to form one long command.

-monochrome +monochrome

Start graph in monochrome (color) mode.

-offList <list>

Comma separated list of line numbers to be initially off. The line numbers begin at 1.

-poll <interval>

Use the specified interval to poll for data. This interval determines the polling frequency of SNMP data. If interval is 0, turn off polling.

-printCommand <command>

This is the command to which the `xwd(1)` compatible graph image will be piped to for printing.

-printGeometry <geometry>

This is the size of the graph created when dumping the graph to a printer. An internal buffer of this size is created, and the resulting image is sent to the `-printCommand` command in an `xwd (1)` format. The size is defined in pixels and should be in the form of `WIDTHxHEIGHT`. The option `-printSize`, below can be used to define the geometry in inches, if this option is not defined.

-printOnly

After graph first displays, print the graph using the print resource, then exit. This feature is very useful for generating alot of graphs in batch mode, such as a cron job once a month to show network statistics for that month. Note that a display is needed to be opened to draw the graph, and a display which is at the VUE login cannot be opened. Also see SIGNALS below.

-printSize <geometry>

This option is the same as `-printGeometry`, except the units are defined in inches instead of pixels.

-resolution <resolution>

If set to 0, the resolution (length of each line segment) is determined by the data set. If set to non-zero, each line segment will be of the specified length. When set to non-zero, each segment will have a maximum and minimum value, which can be displayed by modifying the values column in the View→Line Configuration menu.

-showLegend +showLegend

Initially display (don't display) graph units and legend.

-showGrid +showGrid

Initially display (don't display) graph grid.

-stats +stats

Initially bring up (don't bring up) the statistics window.

-stopTime <string>

The default stopping time for the graph. Normally, this is the end of the data. The format of <string> is the same as the format of time for standard input data. See DATA INPUT EXTENSIBILITY below.

-title <title>

The title for the graph.

-units <units>

The units string graphed in the upper left corner of the graph. The default is no units.

-widthList <list>

Comma separated list of line numbers and their initial width in pixels. The line numbers begin at 1. Each element is in the form *line:width*, such as "1:3,3:3,5:3". Note that the width of the line shown in the View→Line Configuration dialog will show the default width. Reselecting the default width will change the value to the default width. This number will not show up as a line width menu item unless the corresponding line width choices resource *xnmgraph.multValues* also includes this value.

-xAxisFile <file>

This option allows user control of the time labels used across the X-Axis. This is useful if the data you are displaying using the input data extensibility (standard input data) is not really linearly time based. The <file> and standard input file should use time starting at 1 (first stop time at 2), and each new data point is given the next consecutive time. Each line in this file must have a timestamp and a string. Note that fields such as those shown in View→Time Intervals menu will still try to show width in the true time intervals, such as seconds, minutes, and hours. See EXAMPLES below.

RESOURCES

Resources are documented in the */usr/OV/app-defaults/XNm* resource file.

DATA INPUT EXTENSIBILITY

To allow graphing of time based historical data not collected by *snmpCollect(1m)*, the standard input to *xnmgraph* will be used if a mibOID in the line specification begins with a '.', '+', '%', or a '=' character. The stream number is the number which immediately follows this character. Thereafter, data which is received on standard input will be treated as if it had been stored in a data file. Standard input lines beginning with a hash ('#') are comments and are ignored. The stream number must be non-zero.

When using data input extensibility, the nodelist is ignored. If the node in the line specification is defaulted, the mibLabel will be the top level menu item, else it will be cascaded off of nodename.

The form of each line is either

STREAMNUM STARTTIME STOPTIME MIN [AVG] [MAX]

or

MSG <message>

The message is a sequence of characters, with the two characters "\n" ('\' followed by 'n') representing a newline in the message.

In the first form, a data point is added for the corresponding stream number. The stream number represented by STREAMNUM must be defined on the *xnmgraph* command line. Each field is separated by white space. STARTTIME and STOPTIME are the number of seconds since the epoc (time_t value). If STARTTIME is 0, it is assumed to be the same as the last STOPTIME for this data stream, or STOPTIME-1 if this is the first data point. MIN, AVG, and MAX are a floating point numbers, which represents the value of data point in the given time interval. If AVG and MAX are omitted, they default to MIN. Alternatively, STARTTIME and STOPTIME can be given in the form *mm.dd.yy-hh:mm:ss* where *mm* is the month, *dd* is the date, *yy* is the number of years since 1900, *hh* is the hours, *mm* is the minutes, and *ss* is the seconds.

In the second form, a message will be added to the message box. Note that the two characters "\t" are not expanded, only the two characters "\n" are handled specially.

If the mibOID field in the line specification begins with a '+' or '=', then the data is always treated as positive values. If a negative value is encountered, it is converted to the absolute value, and the line is drawn differently. This feature is useful to indicate that the data is somehow questionable or unusual in some

way.

If the mibOID field in the line specification begins with a '%' or '=', then the data is treated as "independent" data streams, and the minimum, average, and maximum values are not required to be numerically related (i.e., the minimum can be greater than the maximum). During data reduction, they will be averaged independently (normally the minimum and maximum values are saved). This is useful if you are using your own input and want the lines to have a meaning beside Minimum, Average, and Maximum. See */usr/OV/app-defaults/XNm* for a discussion on changing the Minimum, Average, and Maximum labels.

EXAMPLES

The following examples demonstrate sample command lines used to graph using *xnmgraph*.

Standard Input Data From a File

The following is an example of using *xnmgraph*'s data input extensibility. If the following lines are in the file *stdinData*, then the command

```
cat stdinData | xnmgraph -mib "-1:....."
could be used to display the data.
```

```
# Start of standard input file: stdinData
1 0 11.05.85-12:00:00 31.00
1 11.05.85-12:00:00 11.05.90-12:00:00 32.00
1 11.05.85-12:00:00 03.19.92-14:58:35 33.00 34 35
# Comments are allowed
1 701042315 701042400 36 37 37.5
MSG End Of Data Set
# End of standard input file: stdinData
```

Standard Input Data From a Shell

To graph random data using the standard input mode of *xnmgraph*, use the following command:

```
awk 'BEGIN{
    for(;;){
        system("sleep 1;date +%m.%d.%y-%H:%M:%S>/tmp/date")
        getline<"/tmp/date";close("/tmp/date")
        printf("1 0 %s %f\n", $0, rand())
    }
}' | xnmgraph -mib "+1:random line:....."
```

If a process is piping data slowly into *xnmgraph* note that the operating system pipes buffer data. This means if your program writes out one line of data per minute to standard input of *xnmgraph*, it may take an hour before enough data is written to flush the data through the pipe. To avoid this problem, add *fflush()* calls to flush the data. If the input program is a script, such as *awk* difficulties may arise, since you cannot *close()* or *fflush()* the data. Using the "+0 -f" option to *tail(1)* within the *awk(1)* script will alleviate this problem when reading from a file. If using *awk(1)*, the command

```
{ printf() | "tail +0 -f" }
```

will cause output data to be non-buffered.

Graphing Live SNMP Data

If you want to graph *ifInOctets* and *ifOutOctets* for instances 2-5 on *node1* and *node2*, use the following command:

```
xnmgraph -mib \
"interfaces.ifTable.ifEntry.ifInOctets:In Bytes:[2-5]:::," \
interfaces.ifTable.ifEntry.ifOutOctets:Out Bytes:[2-5]:::," \
node1 node2
```

If you want to graph *ifInOctets* for all instances on *node1*, using the first part of the interface description as the label (truncated at first space), use the following command:

```
xnmgraph -mib \
".1.3.6.1.2.1.2.1.10:::..1.3.6.1.2.1.2.1.2: :." \
node1
```

If you want to graph *ifInOctets* for all instances on *node1* which are not software loopback (*ifType* = 24) and not other (*ifType* = 1) use the following command: