# Contents

**Preface** ........................................ vii  
Who should read this document ................................ vii  
Prerequisite and related documents ................................ vii  
What this manual contains ................................ viii  
Accessing publications online ................................ viii  
Ordering publications ................................ ix  
Accessibility ................................ ix  
Contacting software support ................................ ix  
Conventions used in this guide ................................ ix  

**Part 1. Platform commands** .................................. 1  

**Chapter 1. Introduction to Tivoli commands** .................. 3  
Establishing the Tivoli environment ................................ 3  
Command syntax ................................ 4  
Object references ................................ 4  
Registered names ................................ 5  
Object paths ................................ 5  
Tivoli transactions ................................ 6  

**Chapter 2. Commands by component** ......................... 7  
Administrator commands ................................ 7  
Configuration management commands ................................ 7  
Endpoint and gateway commands ................................ 7  
httpd commands ................................ 8  
Installation commands ................................ 9  
Interregion commands ................................ 9  
Low-level maintenance commands ................................ 9  
Managed node commands ................................ 10  
Multiplexed distribution commands ................................ 10  
Notification commands ................................ 11  
Policy commands ................................ 11  
Query commands ................................ 12  
RDBMS interface module (RIM) commands .......................... 12  
Revision control system (RCS) commands .......................... 12  
Scheduler commands ................................ 12  
Task library commands ................................ 13  
Miscellaneous commands ................................ 13  

**Chapter 3. Commands alphabetically** ......................... 15  
idlarg ........................................ 16  
idlattr ........................................ 17  
idlcall ........................................ 19  
idlexception .................................... 21  
idlinput ....................................... 23  
idlresult ....................................... 24  
lcf .............................................. 25  
lcfd.sh ......................................... 35  
logs ............................................ 36  
objcall ....................................... 38  
odadmin ....................................... 41  
odbls ......................................... 54  
odstat ....................................... 56  
oinstall ........................................ 61  
oserv .......................................... 63  
tivoli .......................................... 67  
tmcmd .......................................... 69  
tmsstat ........................................ 70  
vdisp .......................................... 72  
w4inslcfl.pl ................................... 73  
waddicon ..................................... 76  
waddpath ..................................... 78  
wadminep ..................................... 79  
wauthadmin ................................... 80  
wbkupdb ....................................... 82  
wbindmsg ....................................... 86  
wbroadcast .................................... 87  
wcatcher ...................................... 88  
wcd ........................................... 90  
wchddep ....................................... 91  
wchkdb ........................................ 92  
wchknode ..................................... 94  
wchkpol ....................................... 96  
wc ............................................... 97  
wclient ....................................... 103  
wcrblkr ....................................... 108  
wcrline ....................................... 109  
wco ........................................... 110  
wconnect ..................................... 115  
wcpdcm ........................................ 118  
wcpyfile ..................................... 120  
wcrtdadmin ................................... 121  
wcrtgate ..................................... 123  
wcrtpgate .................................... 123  
wcrtpol ....................................... 125  
wcrtpol ....................................... 127  
wctpr .......................................... 128  
wctpr .......................................... 129  
wctprfmgr .................................... 131  
wcrqlib ....................................... 132  
wcrquery ..................................... 133  
wcrtrim ....................................... 135  
wcrtask ....................................... 139  
wcrttlib ...................................... 141  
wdate .......................................... 142  
wdel .......................................... 143  
wdelet ....................................... 145  
wdelgate ..................................... 146  
wdeljob ....................................... 147  
wdelpol ....................................... 148  
wdelpr ....................................... 149  
wdeletsched .................................. 150  
wdelet ....................................... 151  
wdepot ....................................... 152  
wdepset ....................................... 155  
wdisconn ..................................... 158  
wdiskspace ................................... 159  
wdist .......................................... 160  
wdisttask .................................... 162  
wdskspec ..................................... 164  

© Copyright IBM Corp. 2003
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdtmsg</td>
<td>165</td>
</tr>
<tr>
<td>weditini</td>
<td>166</td>
</tr>
<tr>
<td>wedsched</td>
<td>167</td>
</tr>
<tr>
<td>wenblsched</td>
<td>170</td>
</tr>
<tr>
<td>wep</td>
<td>171</td>
</tr>
<tr>
<td>wepmgr</td>
<td>181</td>
</tr>
<tr>
<td>wepstatus</td>
<td>186</td>
</tr>
<tr>
<td>wepupgd</td>
<td>189</td>
</tr>
<tr>
<td>wexpnotif</td>
<td>191</td>
</tr>
<tr>
<td>wgateway</td>
<td>192</td>
</tr>
<tr>
<td>wgetadmin</td>
<td>202</td>
</tr>
<tr>
<td>wgetallinst</td>
<td>204</td>
</tr>
<tr>
<td>wgetdfpol</td>
<td>205</td>
</tr>
<tr>
<td>wgeteppol</td>
<td>206</td>
</tr>
<tr>
<td>wgetjob</td>
<td>208</td>
</tr>
<tr>
<td>wgetkey</td>
<td>209</td>
</tr>
<tr>
<td>wgetpolm</td>
<td>210</td>
</tr>
<tr>
<td>wgetpr</td>
<td>212</td>
</tr>
<tr>
<td>wgetprf</td>
<td>213</td>
</tr>
<tr>
<td>wgetquery</td>
<td>214</td>
</tr>
<tr>
<td>wgetrim</td>
<td>215</td>
</tr>
<tr>
<td>wgetsched</td>
<td>216</td>
</tr>
<tr>
<td>wgetsub</td>
<td>218</td>
</tr>
<tr>
<td>wgettask</td>
<td>219</td>
</tr>
<tr>
<td>wgetval</td>
<td>220</td>
</tr>
<tr>
<td>whostid</td>
<td>221</td>
</tr>
<tr>
<td>whttpd</td>
<td>222</td>
</tr>
<tr>
<td>wiconv</td>
<td>224</td>
</tr>
<tr>
<td>wident</td>
<td>225</td>
</tr>
<tr>
<td>widmap</td>
<td>226</td>
</tr>
<tr>
<td>wiconfig</td>
<td>228</td>
</tr>
<tr>
<td>winsblk</td>
<td>230</td>
</tr>
<tr>
<td>winsline</td>
<td>232</td>
</tr>
<tr>
<td>winstall</td>
<td>234</td>
</tr>
<tr>
<td>winstdir</td>
<td>236</td>
</tr>
<tr>
<td>winstendpt</td>
<td>237</td>
</tr>
<tr>
<td>winstclf</td>
<td>238</td>
</tr>
<tr>
<td>winterp</td>
<td>244</td>
</tr>
<tr>
<td>wleftap</td>
<td>245</td>
</tr>
<tr>
<td>wln</td>
<td>247</td>
</tr>
<tr>
<td>wlocalhost</td>
<td>249</td>
</tr>
<tr>
<td>wlocktmr</td>
<td>250</td>
</tr>
<tr>
<td>wlocpath</td>
<td>251</td>
</tr>
<tr>
<td>wlookup</td>
<td>252</td>
</tr>
<tr>
<td>wls</td>
<td>254</td>
</tr>
<tr>
<td>wsconn</td>
<td>255</td>
</tr>
<tr>
<td>wsendpts</td>
<td>257</td>
</tr>
<tr>
<td>wlsinst</td>
<td>258</td>
</tr>
<tr>
<td>wlsnotif</td>
<td>260</td>
</tr>
<tr>
<td>wlspol</td>
<td>262</td>
</tr>
<tr>
<td>wlspolm</td>
<td>263</td>
</tr>
<tr>
<td>wlssub</td>
<td>265</td>
</tr>
<tr>
<td>wlstlib</td>
<td>266</td>
</tr>
<tr>
<td>wmailhost</td>
<td>267</td>
</tr>
<tr>
<td>wmanode</td>
<td>268</td>
</tr>
<tr>
<td>wmcast</td>
<td>269</td>
</tr>
<tr>
<td>wmdist</td>
<td>272</td>
</tr>
<tr>
<td>wmdistgui</td>
<td>281</td>
</tr>
<tr>
<td>wmemsize</td>
<td>282</td>
</tr>
<tr>
<td>wmerge</td>
<td>283</td>
</tr>
<tr>
<td>wmgaei</td>
<td>284</td>
</tr>
<tr>
<td>wmsgini</td>
<td>285</td>
</tr>
<tr>
<td>wmv</td>
<td>286</td>
</tr>
<tr>
<td>wmvrim</td>
<td>287</td>
</tr>
<tr>
<td>wpatch</td>
<td>289</td>
</tr>
<tr>
<td>wping</td>
<td>290</td>
</tr>
<tr>
<td>wpopulate</td>
<td>291</td>
</tr>
<tr>
<td>wpulatepol</td>
<td>292</td>
</tr>
<tr>
<td>wpupdm</td>
<td>293</td>
</tr>
<tr>
<td>wpsd</td>
<td>295</td>
</tr>
<tr>
<td>wrcsm</td>
<td>297</td>
</tr>
<tr>
<td>wrcsidiff</td>
<td>301</td>
</tr>
<tr>
<td>wregmerge</td>
<td>303</td>
</tr>
<tr>
<td>wrefresh</td>
<td>305</td>
</tr>
<tr>
<td>wreregister</td>
<td>306</td>
</tr>
<tr>
<td>wreservers</td>
<td>308</td>
</tr>
<tr>
<td>wrunadms</td>
<td>309</td>
</tr>
<tr>
<td>wrunstart</td>
<td>310</td>
</tr>
<tr>
<td>wrunadms</td>
<td>310</td>
</tr>
<tr>
<td>wrunjob</td>
<td>312</td>
</tr>
<tr>
<td>wrunlog</td>
<td>315</td>
</tr>
<tr>
<td>wrmnode</td>
<td>316</td>
</tr>
<tr>
<td>wrlblb</td>
<td>318</td>
</tr>
<tr>
<td>wrlimension</td>
<td>319</td>
</tr>
<tr>
<td>wrpt</td>
<td>320</td>
</tr>
<tr>
<td>wrunas</td>
<td>326</td>
</tr>
<tr>
<td>wruninvquery</td>
<td>327</td>
</tr>
<tr>
<td>wrunjob</td>
<td>329</td>
</tr>
<tr>
<td>wrunquery</td>
<td>331</td>
</tr>
<tr>
<td>wrunupr</td>
<td>333</td>
</tr>
<tr>
<td>wschedjob</td>
<td>336</td>
</tr>
<tr>
<td>wserver</td>
<td>339</td>
</tr>
<tr>
<td>wsetadmin</td>
<td>345</td>
</tr>
<tr>
<td>wsetdfpol</td>
<td>347</td>
</tr>
<tr>
<td>wseterr</td>
<td>348</td>
</tr>
<tr>
<td>wsetjob</td>
<td>349</td>
</tr>
<tr>
<td>wsetlang</td>
<td>351</td>
</tr>
<tr>
<td>wsetkey</td>
<td>353</td>
</tr>
<tr>
<td>wsetpm</td>
<td>354</td>
</tr>
<tr>
<td>wsetpr</td>
<td>355</td>
</tr>
<tr>
<td>wsetquery</td>
<td>356</td>
</tr>
<tr>
<td>wsetrim</td>
<td>358</td>
</tr>
<tr>
<td>wsetrmpw</td>
<td>360</td>
</tr>
<tr>
<td>wsetap</td>
<td>361</td>
</tr>
<tr>
<td>wsetask</td>
<td>363</td>
</tr>
<tr>
<td>wsetval</td>
<td>364</td>
</tr>
<tr>
<td>wsdnotif</td>
<td>365</td>
</tr>
<tr>
<td>wstandalone</td>
<td>367</td>
</tr>
<tr>
<td>wstartsched</td>
<td>369</td>
</tr>
<tr>
<td>wsub</td>
<td>370</td>
</tr>
<tr>
<td>wsupport</td>
<td>371</td>
</tr>
<tr>
<td>wtainnotif</td>
<td>373</td>
</tr>
<tr>
<td>wtaskabortion</td>
<td>374</td>
</tr>
<tr>
<td>wtimezone</td>
<td>376</td>
</tr>
<tr>
<td>wttemp</td>
<td>378</td>
</tr>
<tr>
<td>wtlr</td>
<td>378</td>
</tr>
<tr>
<td>wtunix</td>
<td>380</td>
</tr>
<tr>
<td>wtrace</td>
<td>381</td>
</tr>
<tr>
<td>wuname</td>
<td>384</td>
</tr>
<tr>
<td>wuninst</td>
<td>385</td>
</tr>
<tr>
<td>wuninstall</td>
<td>387</td>
</tr>
<tr>
<td>wusnjob</td>
<td>389</td>
</tr>
<tr>
<td>wusnsub</td>
<td>391</td>
</tr>
<tr>
<td>wupdate</td>
<td>391</td>
</tr>
</tbody>
</table>
Part 2. Policy methods ........................................ 395

Chapter 4. Endpoint policy scripts ............................ 397

Endpoint policy methods .................................... 397
allow_install_policy ........................................ 398
after_install_policy ........................................ 402
login_policy ............................................... 405
select_gateway_policy ...................................... 407

Chapter 5. Profile manager policy scripts ................. 411

Default policy methods for profile managers ............ 411
Validation policy methods for profile managers ........ 411
Profile manager policy methods ......................... 411
pm_def_profile_managers ................................ 412
pm_def_profile_types .................................... 413
pm_def_subscribers ....................................... 414
pm_val_remove_subscribers ................................ 415
pm_val_remove_subscription ................................ 416
pm_val_subscribers ....................................... 417
pm_val_subscription ....................................... 418

Chapter 6. Task library policy scripts ...................... 419

Default policy methods for task libraries ............... 419
Validation policy methods for task libraries ............ 419
Task library policy methods .............................. 419
tl_def_dist_mode .......................................... 420
tl_def_man_nodes ......................................... 421
tl_def_set_gid ............................................ 422
tl_def_set_uid ............................................ 423
tl_val_man_nodes ......................................... 424
tl_val_prof_mgs ........................................... 425
tl_val_set_gid ............................................. 426
tl_val_set_uid ............................................. 427

Chapter 7. Editing policy scripts ............................ 429

Editing endpoint policy scripts ........................... 429
Editing profile manager and task library policy scripts .................................. 429

Part 3. Task library language ................................ 433

Chapter 8. Task library language overview ................. 435

Using the wtll command .................................. 435
Using message catalogs with a task library ............... 436
Enabling tasks to be run on endpoints ................... 437

Chapter 9. Syntax for the task library language ............ 439

Setting library characteristics ............................ 439
Defining Argument layouts .................................. 441
Text and CryptoText ........................................ 441
Choice and TextChoice ..................................... 443
Example—Choice layout with the Resource keyword .................. 444
Example—Choice layout with the File keyword ................. 446
Example—TextChoice layout with an embedded Choice List ............ 447
ChoiceButton ................................................... 447
Example—Program keyword ................................ 448
Example—Resource keyword ............................... 448
RadioButton .................................................. 449
Example—File keyword .................................... 450
Example—embedded list of radio buttons ................... 450
Creating task definitions .................................. 451
Defining a task header ...................................... 451
Defining an Implementation clause ......................... 452
Defining Argument clauses ................................ 454
Adding comments to a task library ......................... 456
Editing the TLL configuration files ......................... 456

Chapter 10. Sample Task Library ............................. 459

Appendix. Notices .......................................... 479
Trademarks .................................................. 480

Index .......................................................... 483
Preface

Tivoli® Management Framework is the base component for the Tivoli product line. Using Tivoli Management Framework and a combination of Tivoli Enterprise applications, you can manage large distributed networks with multiple operating systems, various network services, diverse system tasks, and constantly changing nodes and users.

Tivoli Management Framework provides a set of common services or features that are used by the Tivoli Enterprise applications installed on Tivoli Management Framework. The services provided by Tivoli Management Framework include, but are not limited to, the following:

- A Dynamic Host Configuration Protocol (DHCP) service that enables dynamic IP addressing for installations that use DHCP
- A task library through which you can create tasks and execute the tasks on multiple Tivoli resources
- A scheduler that enables you to schedule all Tivoli operations, including the execution of tasks created in the Tivoli task library
- A RDBMS Interface Module (RIM) that enables some Tivoli Enterprise applications to write application-specific information to relational databases
- A query facility that enables you to search and retrieve information from a relational database

Tivoli applications installed on Tivoli Management Framework are enabled to use the services provided by Tivoli Management Framework.

This manual provides in-depth information about Tivoli Management Framework commands. This manual is helpful when writing scripts that are later run as Tivoli tasks. This manual also documents Tivoli-provided policy scripts.

References to the interpreter type for a particular client are located throughout this guide. Interpreter types for each machine type are located in the Tivoli Management Framework Release Notes.

Who should read this document

This document is intended for use by system administrators who use the command line to perform Tivoli operations. It is also helpful in writing scripts that are later run as Tivoli tasks. Users of this manual should have some knowledge of the following:

- The UNIX® or Microsoft® Windows® operating systems
- Shell programming
- The Motif or Windows environment

Prerequisite and related documents

Tivoli provides the following related documentation:

- Tivoli Management Framework Planning for Deployment Guide
  - Explains how to plan for deploying your Tivoli environment. It also describes Tivoli Management Framework and its services.
Tivoli Enterprise Installation Guide

Explains how to install and upgrade Tivoli Enterprise™ software within your Tivoli region using the available installation mechanisms provided by Tivoli Software Installation Service and Tivoli Management Framework. Tivoli Enterprise software includes the Tivoli server, managed nodes, gateways, endpoints, and RDBMS Interface Module (RIM) objects. This guide also provides information about troubleshooting installation problems.

Tivoli Management Framework User’s Guide

Describes the concepts and procedures for using Tivoli Management Framework services. It provides instructions for performing tasks from the Tivoli desktop and from the command line.

Tivoli Management Framework Maintenance and Troubleshooting Guide

Explains how to maintain a Tivoli environment and troubleshoot problems that can arise during normal operations.

What this manual contains

The Tivoli Management Framework Reference Manual contains the following sections:

- Part 1, “Platform commands,” on page 1
  Provides in-depth information about Tivoli commands provided by Tivoli Management Framework.

  Tivoli commands enable you to perform system operations from a UNIX command line instead of using the Tivoli desktop. This is often useful when you do not have access to a graphical display such as when logging in over a modem line.

- Part 2, “Policy methods,” on page 395
  Provides in-depth information about the default and validation policy scripts and methods provided by Tivoli Management Framework.

  Tivoli Management Framework uses default and validation policy in the task library and profile manager. These services use default and validation methods that call shell scripts to set or validate data. You can edit any of the shell scripts to create custom policies for your organization.

  Tivoli gateways and endpoint managers also use policy scripts. The endpoint scripts differ from the default and validation policy in that policy objects are associated with the endpoint scripts.

- Part 3, “Task library language,” on page 433
  Provides in-depth information about creating task libraries using the Tivoli Task Library Language.

Accessing publications online

The documentation CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli software information center Web site. Access the Tivoli software information center by first going to the Tivoli software library at the following Web address:

http://publib.boulder.ibm.com/tividd/td/tdprodlist.html
Note: If you print PDF documents on other than letter-sized paper, set the option in the File → Print window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com

From this Web page, select Publications and follow the instructions.

You can also order by telephone by calling one of these numbers:

• In the United States: 800-879-2755
• In Canada: 800-426-4968

In other countries, see the following Web site for a list of telephone numbers:

http://www.ibm.com/software/tivoli/order-lit

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

Contacting software support

If you have a problem with any Tivoli product, refer to the following IBM Software Support Web site:


If you want to contact software support, see the IBM Software Support Guide at the following Web site:

http://techsupport.services.ibm.com/guides/handbook.html

The guide provides information about how to contact IBM Software Support, depending on the severity of your problem, and the following information:

• Registration and eligibility
• Telephone numbers, depending on the country in which you are located
• Information you must have before contacting IBM Software Support

Conventions used in this guide

This guide uses the following typeface conventions:

Bold

• Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
• Interface controls
• Keywords and parameters in text

Italic
• Words defined in text
• Emphasis of words (words as words)
• New terms in text (except in a definition list)
• Variables and values you must provide

Monospace
• Examples and code examples
• Message text and prompts addressed to the user
• Text that the user must type
• Values for arguments or command options

This guide uses the UNIX convention for specifying environment variables and for directory notation:
• When using the Windows command line, replace $variable with %variable% for environment variables and replace each forward slash (/) with a backslash (\) in directory paths.
• When using the bash shell on Windows operating systems, use the UNIX conventions.
# Part 1. Platform commands

## Chapter 1. Introduction to Tivoli commands

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establishing the Tivoli environment</td>
<td>3</td>
</tr>
<tr>
<td>Command syntax</td>
<td>4</td>
</tr>
<tr>
<td>Object references</td>
<td>5</td>
</tr>
<tr>
<td>Registered names</td>
<td>5</td>
</tr>
<tr>
<td>Object paths</td>
<td>6</td>
</tr>
</tbody>
</table>

## Chapter 2. Commands by component

<table>
<thead>
<tr>
<th>Component</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator commands</td>
<td>7</td>
</tr>
<tr>
<td>Configuration management commands</td>
<td>7</td>
</tr>
<tr>
<td>Endpoint and gateway commands</td>
<td>7</td>
</tr>
<tr>
<td>httpd commands</td>
<td>8</td>
</tr>
<tr>
<td>Installation commands</td>
<td>9</td>
</tr>
<tr>
<td>Interregion commands</td>
<td>9</td>
</tr>
<tr>
<td>Low-level maintenance commands</td>
<td>9</td>
</tr>
<tr>
<td>Managed node commands</td>
<td>10</td>
</tr>
<tr>
<td>Multiplexed distribution commands</td>
<td>10</td>
</tr>
<tr>
<td>Notification commands</td>
<td>11</td>
</tr>
<tr>
<td>Policy commands</td>
<td>11</td>
</tr>
<tr>
<td>Query commands</td>
<td>12</td>
</tr>
<tr>
<td>RDBMS interface module (RIM) commands</td>
<td>12</td>
</tr>
<tr>
<td>Revision control system (RCS) commands</td>
<td>12</td>
</tr>
<tr>
<td>Scheduler commands</td>
<td>12</td>
</tr>
<tr>
<td>Task library commands</td>
<td>13</td>
</tr>
<tr>
<td>Miscellaneous commands</td>
<td>13</td>
</tr>
</tbody>
</table>

## Chapter 3. Commands alphabetically

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>idlarg</td>
<td>15</td>
</tr>
<tr>
<td>idlattr</td>
<td>16</td>
</tr>
<tr>
<td>idcall</td>
<td>17</td>
</tr>
<tr>
<td>idexception</td>
<td>19</td>
</tr>
<tr>
<td>idinput</td>
<td>21</td>
</tr>
<tr>
<td>idresult</td>
<td>23</td>
</tr>
<tr>
<td>lcfd</td>
<td>25</td>
</tr>
<tr>
<td>lcfd.sh</td>
<td>35</td>
</tr>
<tr>
<td>logls</td>
<td>36</td>
</tr>
<tr>
<td>objcall</td>
<td>38</td>
</tr>
<tr>
<td>odadmin</td>
<td>41</td>
</tr>
<tr>
<td>odbls</td>
<td>54</td>
</tr>
<tr>
<td>odstat</td>
<td>56</td>
</tr>
<tr>
<td>oinstall</td>
<td>61</td>
</tr>
<tr>
<td>oserv</td>
<td>63</td>
</tr>
<tr>
<td>tivoli</td>
<td>67</td>
</tr>
<tr>
<td>tmcmd</td>
<td>69</td>
</tr>
<tr>
<td>tmstat</td>
<td>70</td>
</tr>
<tr>
<td>vdisp</td>
<td>72</td>
</tr>
<tr>
<td>w4inslcfl.pl</td>
<td>73</td>
</tr>
<tr>
<td>waddicon</td>
<td>76</td>
</tr>
<tr>
<td>waddpath</td>
<td>78</td>
</tr>
<tr>
<td>wadminep</td>
<td>79</td>
</tr>
<tr>
<td>wauthadmin</td>
<td>80</td>
</tr>
<tr>
<td>wbkupdb</td>
<td>82</td>
</tr>
<tr>
<td>wbndmsg</td>
<td>86</td>
</tr>
<tr>
<td>wbroadcast</td>
<td>87</td>
</tr>
<tr>
<td>wcather</td>
<td>88</td>
</tr>
<tr>
<td>wcd</td>
<td>90</td>
</tr>
<tr>
<td>wchdep</td>
<td>91</td>
</tr>
<tr>
<td>wchkdb</td>
<td>92</td>
</tr>
<tr>
<td>wchnode</td>
<td>94</td>
</tr>
<tr>
<td>wchkpol</td>
<td>96</td>
</tr>
<tr>
<td>wci</td>
<td>97</td>
</tr>
<tr>
<td>wclient</td>
<td>103</td>
</tr>
<tr>
<td>wcrblk</td>
<td>108</td>
</tr>
<tr>
<td>wcrline</td>
<td>109</td>
</tr>
<tr>
<td>wco</td>
<td>110</td>
</tr>
<tr>
<td>wconnect</td>
<td>115</td>
</tr>
<tr>
<td>wpcdrom</td>
<td>118</td>
</tr>
<tr>
<td>wpcpyfile</td>
<td>120</td>
</tr>
<tr>
<td>wrctadmin</td>
<td>121</td>
</tr>
<tr>
<td>wrctgate</td>
<td>123</td>
</tr>
<tr>
<td>wrctjob</td>
<td>125</td>
</tr>
<tr>
<td>wrctp</td>
<td>127</td>
</tr>
<tr>
<td>wrctpr</td>
<td>128</td>
</tr>
<tr>
<td>wrctprf</td>
<td>129</td>
</tr>
<tr>
<td>wrctprfmgr</td>
<td>131</td>
</tr>
<tr>
<td>wrctqlib</td>
<td>132</td>
</tr>
<tr>
<td>wrctquery</td>
<td>133</td>
</tr>
<tr>
<td>wrctrim</td>
<td>135</td>
</tr>
<tr>
<td>wrcttask</td>
<td>139</td>
</tr>
<tr>
<td>wrctlit</td>
<td>141</td>
</tr>
<tr>
<td>wdate</td>
<td>142</td>
</tr>
<tr>
<td>wdel</td>
<td>143</td>
</tr>
<tr>
<td>wdelcap</td>
<td>145</td>
</tr>
<tr>
<td>wdelgate</td>
<td>146</td>
</tr>
<tr>
<td>wdeljob</td>
<td>147</td>
</tr>
<tr>
<td>wdelpol</td>
<td>148</td>
</tr>
<tr>
<td>wdelpr</td>
<td>149</td>
</tr>
<tr>
<td>wdesched</td>
<td>150</td>
</tr>
<tr>
<td>wdeltask</td>
<td>151</td>
</tr>
<tr>
<td>wdepot</td>
<td>152</td>
</tr>
<tr>
<td>wdepset</td>
<td>155</td>
</tr>
<tr>
<td>wdisconn</td>
<td>158</td>
</tr>
<tr>
<td>wdiskspace</td>
<td>159</td>
</tr>
<tr>
<td>wdistrib</td>
<td>160</td>
</tr>
<tr>
<td>wdistask</td>
<td>162</td>
</tr>
<tr>
<td>wdskspec</td>
<td>164</td>
</tr>
<tr>
<td>wdtmsg</td>
<td>165</td>
</tr>
<tr>
<td>weditini</td>
<td>166</td>
</tr>
<tr>
<td>wedsched</td>
<td>167</td>
</tr>
<tr>
<td>wenbised</td>
<td>170</td>
</tr>
<tr>
<td>wem</td>
<td>171</td>
</tr>
<tr>
<td>wepmgr</td>
<td>181</td>
</tr>
<tr>
<td>wepstatus</td>
<td>186</td>
</tr>
<tr>
<td>wepupgd</td>
<td>189</td>
</tr>
<tr>
<td>weppnift</td>
<td>191</td>
</tr>
<tr>
<td>wgateway</td>
<td>192</td>
</tr>
<tr>
<td>wegetadmin</td>
<td>202</td>
</tr>
<tr>
<td>wegetallinst</td>
<td>204</td>
</tr>
<tr>
<td>wegetdpol</td>
<td>205</td>
</tr>
<tr>
<td>wegetepol</td>
<td>206</td>
</tr>
<tr>
<td>wegetjob</td>
<td>208</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>wgetkey</td>
<td>209</td>
</tr>
<tr>
<td>wgetpolm</td>
<td>210</td>
</tr>
<tr>
<td>wgetpr</td>
<td>212</td>
</tr>
<tr>
<td>wgetprf</td>
<td>213</td>
</tr>
<tr>
<td>wgetquery</td>
<td>214</td>
</tr>
<tr>
<td>wgettrim</td>
<td>215</td>
</tr>
<tr>
<td>wgetsched</td>
<td>216</td>
</tr>
<tr>
<td>wgetsub</td>
<td>218</td>
</tr>
<tr>
<td>wgettask</td>
<td>219</td>
</tr>
<tr>
<td>wgetval</td>
<td>220</td>
</tr>
<tr>
<td>whostid</td>
<td>221</td>
</tr>
<tr>
<td>whiptd</td>
<td>222</td>
</tr>
<tr>
<td>wiconv</td>
<td>224</td>
</tr>
<tr>
<td>wident</td>
<td>225</td>
</tr>
<tr>
<td>widmap</td>
<td>226</td>
</tr>
<tr>
<td>wifconfig</td>
<td>228</td>
</tr>
<tr>
<td>winsblk</td>
<td>230</td>
</tr>
<tr>
<td>winsline</td>
<td>232</td>
</tr>
<tr>
<td>wininstall</td>
<td>234</td>
</tr>
<tr>
<td>winsdir</td>
<td>236</td>
</tr>
<tr>
<td>winstendpt</td>
<td>237</td>
</tr>
<tr>
<td>winstclf</td>
<td>238</td>
</tr>
<tr>
<td>winterop</td>
<td>244</td>
</tr>
<tr>
<td>wcftap</td>
<td>245</td>
</tr>
<tr>
<td>wln</td>
<td>247</td>
</tr>
<tr>
<td>wlocalhost</td>
<td>249</td>
</tr>
<tr>
<td>wlockmr</td>
<td>250</td>
</tr>
<tr>
<td>wlocpath</td>
<td>251</td>
</tr>
<tr>
<td>wlookup</td>
<td>252</td>
</tr>
<tr>
<td>wls</td>
<td>254</td>
</tr>
<tr>
<td>wlsconn</td>
<td>255</td>
</tr>
<tr>
<td>wlsendpts</td>
<td>257</td>
</tr>
<tr>
<td>wlsinst</td>
<td>258</td>
</tr>
<tr>
<td>wlsnotif</td>
<td>260</td>
</tr>
<tr>
<td>wlspol</td>
<td>262</td>
</tr>
<tr>
<td>wlspolm</td>
<td>263</td>
</tr>
<tr>
<td>wlssub</td>
<td>265</td>
</tr>
<tr>
<td>wlstlib</td>
<td>266</td>
</tr>
<tr>
<td>wmailhost</td>
<td>267</td>
</tr>
<tr>
<td>wmanode</td>
<td>268</td>
</tr>
<tr>
<td>wmcast</td>
<td>269</td>
</tr>
<tr>
<td>wmdist</td>
<td>272</td>
</tr>
<tr>
<td>wmdistgui</td>
<td>281</td>
</tr>
<tr>
<td>wmemsizeln</td>
<td>282</td>
</tr>
<tr>
<td>wmerge</td>
<td>283</td>
</tr>
<tr>
<td>wmrgaef</td>
<td>284</td>
</tr>
<tr>
<td>wmrgini</td>
<td>285</td>
</tr>
<tr>
<td>wmv</td>
<td>286</td>
</tr>
<tr>
<td>wmvrim</td>
<td>287</td>
</tr>
<tr>
<td>wpatch</td>
<td>289</td>
</tr>
<tr>
<td>wping</td>
<td>290</td>
</tr>
<tr>
<td>wpopulate</td>
<td>291</td>
</tr>
<tr>
<td>wputeppol</td>
<td>292</td>
</tr>
<tr>
<td>wputpolm</td>
<td>293</td>
</tr>
<tr>
<td>wpwd</td>
<td>296</td>
</tr>
<tr>
<td>wrs</td>
<td>297</td>
</tr>
<tr>
<td>wrcsdiff</td>
<td>301</td>
</tr>
<tr>
<td>wrcsmmerge</td>
<td>303</td>
</tr>
<tr>
<td>wrefresh</td>
<td>305</td>
</tr>
<tr>
<td>wregister</td>
<td>306</td>
</tr>
<tr>
<td>wrestart</td>
<td>308</td>
</tr>
<tr>
<td>wrimtest</td>
<td>309</td>
</tr>
<tr>
<td>wrimtrace</td>
<td>310</td>
</tr>
<tr>
<td>wrlog</td>
<td>312</td>
</tr>
<tr>
<td>wrm</td>
<td>315</td>
</tr>
<tr>
<td>wrmnode</td>
<td>316</td>
</tr>
<tr>
<td>wrplibk</td>
<td>318</td>
</tr>
<tr>
<td>wrpline</td>
<td>319</td>
</tr>
<tr>
<td>wrpt</td>
<td>320</td>
</tr>
<tr>
<td>wrunas</td>
<td>326</td>
</tr>
<tr>
<td>wruninvquery</td>
<td>327</td>
</tr>
<tr>
<td>wrunjob</td>
<td>329</td>
</tr>
<tr>
<td>wrunquery</td>
<td>331</td>
</tr>
<tr>
<td>wruntask</td>
<td>333</td>
</tr>
<tr>
<td>wschedjob</td>
<td>336</td>
</tr>
<tr>
<td>wserver</td>
<td>339</td>
</tr>
<tr>
<td>wsetadmin</td>
<td>345</td>
</tr>
<tr>
<td>wsetdfpol</td>
<td>347</td>
</tr>
<tr>
<td>wseterr</td>
<td>348</td>
</tr>
<tr>
<td>wsetjob</td>
<td>349</td>
</tr>
<tr>
<td>wsetlang</td>
<td>351</td>
</tr>
<tr>
<td>wsetpkey</td>
<td>353</td>
</tr>
<tr>
<td>wsetpm</td>
<td>354</td>
</tr>
<tr>
<td>wsetpr</td>
<td>355</td>
</tr>
<tr>
<td>wsetquery</td>
<td>356</td>
</tr>
<tr>
<td>wsetrim</td>
<td>358</td>
</tr>
<tr>
<td>wsetrimpw</td>
<td>360</td>
</tr>
<tr>
<td>wsettap</td>
<td>361</td>
</tr>
<tr>
<td>wsettask</td>
<td>363</td>
</tr>
<tr>
<td>wsetval</td>
<td>364</td>
</tr>
<tr>
<td>wsendnotif</td>
<td>365</td>
</tr>
<tr>
<td>wstandalone</td>
<td>367</td>
</tr>
<tr>
<td>wstartsched</td>
<td>369</td>
</tr>
<tr>
<td>wsub</td>
<td>370</td>
</tr>
<tr>
<td>wsupport</td>
<td>371</td>
</tr>
<tr>
<td>wtailnotif</td>
<td>373</td>
</tr>
<tr>
<td>wtaskabort</td>
<td>374</td>
</tr>
<tr>
<td>wtimezone</td>
<td>376</td>
</tr>
<tr>
<td>wttemp</td>
<td>377</td>
</tr>
<tr>
<td>wtll</td>
<td>378</td>
</tr>
<tr>
<td>wtlnname</td>
<td>380</td>
</tr>
<tr>
<td>wtrace</td>
<td>381</td>
</tr>
<tr>
<td>wubname</td>
<td>384</td>
</tr>
<tr>
<td>wuninst</td>
<td>385</td>
</tr>
<tr>
<td>wunstmr</td>
<td>387</td>
</tr>
<tr>
<td>wunsub</td>
<td>389</td>
</tr>
<tr>
<td>wupdate</td>
<td>391</td>
</tr>
<tr>
<td>wvalidate</td>
<td>392</td>
</tr>
<tr>
<td>wxterm</td>
<td>393</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction to Tivoli commands

Tivoli commands enable you to perform system operations from a UNIX command line instead of using the Tivoli desktop. This is often useful when you do not have access to a graphical display such as when logging in over a modem line.

All Tivoli end-user commands begin with a w to identify them as Tivoli commands. Commands are also developed with a w+verb+object syntax, which matches the way you would think of the action. For example, if you want to create a task, use the wcrttask command. To delete a job, use the wdeljob command.

It is often convenient or more appropriate to invoke a Tivoli management application operation from the command line than from the graphical user interface. For example:
• You may not have access to a graphical user interface, perhaps because you dialed in over a modem
• A number of operations are going to be grouped together inside a shell script
• You would rather invoke a command from a shell

Establishing the Tivoli environment

Tivoli Management Framework includes the setup_env.sh and setup_env.csh scripts, which enable you to establish the correct search paths and environment variables. These setup scripts are available on any server or managed node in the Tivoli region.

To set up the Tivoli environment variables, follow these steps:
1. Log in to a managed node or the Tivoli server.
2. Do one of the following:
   On UNIX operating systems only, do one of the following:
   • If you are using the Bourne shell, enter the following script:
     . /etc/Tivoli/setup_env.sh
   • If you are using the C shell, enter the following script:
     source /etc/Tivoli/setup_env.csh

   On Windows operating systems only, do one of the following:
   • From a bash shell, enter the following command:
     %SystemRoot%\system32\drivers\etc\Tivoli\setup_env.sh
   • From a command prompt, enter the following command:
     %SystemRoot%\system32\drivers\etc\Tivoli\setup_env.cmd
   • To configure the Windows command line to automatically source the Tivoli environment, follow these steps:
     a. Right-click the Command Prompt (MS-DOS) shortcut to the Tivoli desktop.
     b. Click Properties.
     c. Click the Shortcut tab.
     d. In the Target text box, enter the following commands on a single line:
Command syntax

The commands in this book use the following special characters to define the command syntax:

[ ] Identifies optional options. Options not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous option.

[ ] Indicates mutually exclusive information. You can use the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use of the command.

{} Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([ ]).

\ Indicates that the command line wraps to the next line. It is a continuation character.

For example:

logls [-Dofls] [-k dir] [-m maxdlen] log_name...

The log_name option is the only required option for the logls command. The ellipsis following the log_name option indicate that you can specify multiple log file names. The brackets around the other options indicate that these options are optional.

Another example is the wchkdb command:

wchkdb [-o outfile] [-u] [-x] [-f infile | -i | object...]

In this example, the -f infile, -i, and object options are mutually exclusive. The braces ([ ]) indicate that one of these options is required. If you choose to specify the object option, you can optionally specify more than one object name or ID.

The options for each command are listed alphabetically in the Options section, unless the options must be used in a specific order to implement the command.

Object references

When an object is referenced in a command issued from the command line, the reference is not an absolute object reference like those used in programming. Instead, a user-friendly name is used. This user-friendly name derives from a name given to the object by the user of the application, such as when a policy region is created.

Two different forms of names can be used with commands:

• Registered names
• Object paths

Tivoli programs that use a command line interface support both naming schemes. Sometimes you will find it more convenient to use one form over the other. If you receive an error message indicating that a resource cannot be found, try a different naming convention.
Registered names

The key concept behind the name registry is a registered name. A registered name is the name by which a resource instance is registered with the name registry when it is created. Every resource has a name and is of some particular type. For example, a printer named lp01 has a name lp01 and is of type printer. Some examples of registered names used as options for the wls and wmv commands are as follows:

wls @PolicyRegion:Servers
wmv @ManagedNode:ayers-rock @PolicyRegion:Servers

The syntax for specifying a resource using the registered name facility is @type:name, where type is the resource type and name is the particular instance of that resource on which you want to perform some operation.

The name registry does not allow two resources of the same type to have the same name within a single Tivoli region. However, it is possible for resource names to be duplicated within two or more connected regions. If you attempt to perform an action on a resource with a duplicated name, an error message is returned, and the action is not performed.

To avoid this situation, you should either rename one of the resources or differentiate between the resources by appending a region name to the resource name, as follows:

wls @ManagedNode:moria#moria-Region

Object paths

Object paths are similar to path names in file systems and may be relative or absolute. An absolute path is one that starts with a slash (/) character. A relative path can start with any character including the special path components period (.) and double period (..). Some examples of object path names used as options for the wls and wmv commands are as follows:

wls /Regions/Servers
wmv ../Servers/ayers-rock /Regions/Servers

The syntax for specifying a resource using the object path name style is /distinguished/parent/[type:]name, where distinguished is a resource type, parent is the start of the object path name, type is used to further identify a resource, and name is the particular instance on which you want to perform some operation. You often use the optional type qualifier when you need to name a particular resource that has the same name as some other resource of a different type.

For example, suppose policy region Engineering had a profile manager named Servers and a policy subregion named Servers. To specify the profile manager using an object path name, you could use the following:

wls /Regions/Engineering/ProfileManager:Servers

If you specify a resource using an absolute path, its location is not ambiguous between connected regions. However, if you use a relative path, both your home and current administrator collection must be located before the resource can be found. The home collection for each administrator is /Administrators/name, where name is the name of the Tivoli administrator.

If you have recently issued a wcd command, Tivoli Management Framework contains a record that specifies the location of the current administrator collection. Otherwise, no such record exists, and in this case, the current administrator collection can be ambiguous if multiple Tivoli regions are connected. For example,
suppose you are an administrator named John (with a login name john)c in region A, and there is another administrator named John (with a login name of jsmith) in region B. When you specify an action to be performed on a resource, Tivoli Management Framework searches for the /Administrators/John collection. The search finds collections belonging to you and jsmith. Because Tivoli Management Framework cannot determine which home collection you meant to specify, an error message is returned, and the action is not performed. You can execute the wcd command to prevent this problem from occurring.

Tivoli transactions

Because Tivoli products run in a distributed environment, program error conditions can cause consistency errors between the Tivoli object database and the actual work environment. For example, if a Tivoli Enterprise application creates a new user on a managed node, that application must create a new user object and add an entry to the node password file. If an error occurs while writing the file, the application must have a way to remove the user object. Otherwise, the Tivoli database is left in an inconsistent state.

To solve this problem, Tivoli provides a transaction system. A transaction is a set of operations that must all complete successfully. If any operations in a transaction fail, all changes made by the other operations are aborted.

The Tivoli transaction allows you to create nested transactions, which form transaction hierarchies. A method can be invoked in one of the following ways:

- The method is not part of a transaction.
- The method is the top of a transaction hierarchy.
- The method is a subtransaction of a top-level transaction.
- The method is a revocable subtransaction.

A top-level transaction only succeeds if its subtransactions succeed. If the top-level transaction fails, Tivoli undoes all changes made by the transaction and its subtransactions. Subtransactions can themselves have subtransactions on which they depend.

When a subtransaction is revocable, it can abort without forcing the parent transaction to abort. The parent can decide whether the failure is severe enough to warrant aborting the entire transaction. For example, consider a transaction that calls a revocable subtransaction to write a set of database records. If the subtransaction fails after writing 99 percent of the records, the parent may choose to succeed anyway. However, if the method fails after writing only 5 percent, the parent may choose to fail and undo the records that have been written.

The Tivoli task library runs jobs using this transaction model. Tasks run from the Tivoli desktop are executed as top-level transactions. You can, however, create a task that runs a shell script, which in turn runs other Tivoli tasks as either subtransactions or as revocable subtransactions.
Chapter 2. Commands by component

The following sections list Tivoli Management Framework commands by component. Each section contains a table of command names and purpose statements.

**Administrator commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wauthadmin</td>
<td>Adds, removes, or displays the root authority of Tivoli administrators in a Tivoli region.</td>
</tr>
<tr>
<td>wcrtadmin</td>
<td>Creates a new Tivoli administrator.</td>
</tr>
<tr>
<td>wgetadmin</td>
<td>Lists information about a Tivoli administrator.</td>
</tr>
<tr>
<td>widmap</td>
<td>Lists and modifies user login mapping.</td>
</tr>
<tr>
<td>wsetadmin</td>
<td>Changes information about a Tivoli administrator.</td>
</tr>
<tr>
<td>wsetlang</td>
<td>Determines the operating system locale to use for a Tivoli server or managed node.</td>
</tr>
</tbody>
</table>

**Configuration management commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcrtprf</td>
<td>Creates a new profile or clones an existing profile.</td>
</tr>
<tr>
<td>wcrtprfmgr</td>
<td>Creates a profile manager.</td>
</tr>
<tr>
<td>wdistrib</td>
<td>Distributes one or more profile copies.</td>
</tr>
<tr>
<td>wgetprf</td>
<td>Retrieves subscription copies of one or more profiles.</td>
</tr>
<tr>
<td>wgetsub</td>
<td>Lists the subscribes of a profile manager.</td>
</tr>
<tr>
<td>wlssub</td>
<td>Lists the profile managers to which a host, NIS domain, or profile manager subscribes.</td>
</tr>
<tr>
<td>wpopulate</td>
<td>Populates a profile from system files.</td>
</tr>
<tr>
<td>wsetpm</td>
<td>Enables or disables a profile manager to operate in dataless mode.</td>
</tr>
<tr>
<td>wsub</td>
<td>Subscribes Tivoli resources to a profile manager.</td>
</tr>
<tr>
<td>wuninst</td>
<td>Uninstalls Tivoli applications from a specified node or from the entire Tivoli region.</td>
</tr>
<tr>
<td>wunsub</td>
<td>Removes Tivoli resources from the subscription list of a profile manager.</td>
</tr>
<tr>
<td>wvalidate</td>
<td>Validates a profile against its validation policy.</td>
</tr>
</tbody>
</table>

**Endpoint and gateway commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcfd</td>
<td>Starts the endpoint daemon (lcfd) on an endpoint and installs or removes the daemon as a service on Windows operating systems.</td>
</tr>
<tr>
<td>Command</td>
<td>Purpose</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lcfd.sh</td>
<td>Starts or stops the endpoint daemon (lcfd) on UNIX and Linux endpoints.</td>
</tr>
<tr>
<td>w4inslcf.pl</td>
<td>Installs an AS/400 endpoint on an iSeries system.</td>
</tr>
<tr>
<td>wadddpath</td>
<td>Adds an entry to the path statement in the registry hive of the current control set. (Windows only)</td>
</tr>
<tr>
<td>wadminep</td>
<td>Automatically upgrades an endpoint.</td>
</tr>
<tr>
<td>wcrlblk</td>
<td>Removes a block of statements from a file.</td>
</tr>
<tr>
<td>wcrline</td>
<td>Removes a single line from a file.</td>
</tr>
<tr>
<td>wcypfile</td>
<td>Enables an .NCF configuration program to copy a file. (NetWare only)</td>
</tr>
<tr>
<td>wcrtgate</td>
<td>Creates a gateway.</td>
</tr>
<tr>
<td>wdelep</td>
<td>Deletes an endpoint.</td>
</tr>
<tr>
<td>wdelgate</td>
<td>Deletes a gateway.</td>
</tr>
<tr>
<td>wdskspc</td>
<td>Verifies the amount of disk space available. (Windows and NetWare only)</td>
</tr>
<tr>
<td>weditini</td>
<td>Modifies the groups, variables, and values in an .ini file</td>
</tr>
<tr>
<td>wep</td>
<td>Performs actions on endpoint information contained in the endpoint list.</td>
</tr>
<tr>
<td>wepmgr</td>
<td>Provides control and configuration for the endpoint manager.</td>
</tr>
<tr>
<td>wepstatus</td>
<td>Returns the status of endpoints.</td>
</tr>
<tr>
<td>wepupgd</td>
<td>Upgrades an endpoint to the newest software.</td>
</tr>
<tr>
<td>Note: Replaces the wadminep upgrade command.</td>
<td></td>
</tr>
<tr>
<td>wgateway</td>
<td>Starts, stops, or lists the properties of an endpoint gateway.</td>
</tr>
<tr>
<td>wgetkey</td>
<td>Retrieves the subkey listing in a registry hive. (Windows only)</td>
</tr>
<tr>
<td>wgetval</td>
<td>Retrieves a registry subkey. (Windows only)</td>
</tr>
<tr>
<td>winsblk</td>
<td>Inserts a block of statements into a file.</td>
</tr>
<tr>
<td>winsline</td>
<td>Inserts a single line into a file.</td>
</tr>
<tr>
<td>winslcf</td>
<td>Installs an endpoint. (Linux, UNIX, and Windows only)</td>
</tr>
<tr>
<td>wlsendpts</td>
<td>Lists all the endpoints subscribed to a profile manager.</td>
</tr>
<tr>
<td>wmrnini</td>
<td>Merges groups and variables from one .INI file into another.</td>
</tr>
<tr>
<td>wrestart</td>
<td>Initiates a system restart and optional restart. (Windows only)</td>
</tr>
<tr>
<td>wrplblk</td>
<td>Replaces a block of statements in a file.</td>
</tr>
<tr>
<td>wrplline</td>
<td>Replaces a single line in a file.</td>
</tr>
<tr>
<td>wseterr</td>
<td>Sets the return code from a batch file for a configuration program.</td>
</tr>
<tr>
<td>wsetval</td>
<td>Sets a registry key value. (Windows only)</td>
</tr>
<tr>
<td>wstandalone</td>
<td>Installs a managed node or endpoint without using rsh or reexec.</td>
</tr>
</tbody>
</table>

**httpd commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>whttpd</td>
<td>Uninstalls the Tivoli HTTP service or forwards HTTP request to a third-party HTTP server.</td>
</tr>
</tbody>
</table>
## Installation commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>oinstall</td>
<td>Installs, updates, or removes the Tivoli object dispatcher service in the Windows Service Manager.</td>
</tr>
<tr>
<td>wclient</td>
<td>Installs Tivoli clients.</td>
</tr>
<tr>
<td>wcpdrom</td>
<td>Copies installation images from a CD to a system directory.</td>
</tr>
<tr>
<td>winstall</td>
<td>Installs a Tivoli Enterprise product.</td>
</tr>
<tr>
<td>winstlcf</td>
<td>Installs an endpoint on a Linux, UNIX, and Windows workstation.</td>
</tr>
<tr>
<td>wmailhost</td>
<td>Specifies the mail server used by Windows managed nodes.</td>
</tr>
<tr>
<td>wpatch</td>
<td>Installs a Tivoli patch.</td>
</tr>
<tr>
<td>wserver</td>
<td>Installs the Tivoli server. (Linux and UNIX only)</td>
</tr>
<tr>
<td>wsettap</td>
<td>Sets the properties of the Tivoli Authentication Package. (Windows only)</td>
</tr>
<tr>
<td>wstandalone</td>
<td>Installs a managed node or endpoint without using rsh or reexec.</td>
</tr>
</tbody>
</table>

## Interregion commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wconnect</td>
<td>Connects two Tivoli regions.</td>
</tr>
<tr>
<td>wdisconnect</td>
<td>Disconnects two Tivoli regions.</td>
</tr>
<tr>
<td>wlookup</td>
<td>Searches for the object reference of a resource.</td>
</tr>
<tr>
<td>wlsconn</td>
<td>Lists the current Tivoli region connections or information about a single connection.</td>
</tr>
<tr>
<td>wregister</td>
<td>Registers a resource with the name registry.</td>
</tr>
<tr>
<td>wtmmname</td>
<td>Displays or changes the name of the local Tivoli region.</td>
</tr>
<tr>
<td>wupdate</td>
<td>Updates resources in the local name registry.</td>
</tr>
</tbody>
</table>

## Low-level maintenance commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>idlarg</td>
<td>Extracts individual options from an option list returned by the idlinput command.</td>
</tr>
<tr>
<td>idlattr</td>
<td>Gets or sets implementation attributes.</td>
</tr>
<tr>
<td>idcall</td>
<td>Provides a method of invoking Interface Definition Language (IDL) operations from the shell command line</td>
</tr>
<tr>
<td>idlexception</td>
<td>Raises exceptions for a shell method.</td>
</tr>
<tr>
<td>idlinput</td>
<td>Gets the input or inout options list to a shell method.</td>
</tr>
<tr>
<td>idlresult</td>
<td>Formats inout or output options or the result (if any) of a shell method.</td>
</tr>
<tr>
<td>logls</td>
<td>Creates a readable version of a transaction log file.</td>
</tr>
<tr>
<td>objcall</td>
<td>Performs an object call from the shell.</td>
</tr>
<tr>
<td>Command</td>
<td>Purpose</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>odadmin</td>
<td>Manages object dispatchers.</td>
</tr>
<tr>
<td>odblsl</td>
<td>Lists the contents of an object database.</td>
</tr>
<tr>
<td>odstat</td>
<td>Lists the status of current and recent object calls.</td>
</tr>
<tr>
<td>oserv</td>
<td>Provides operations to control and configure object dispatchers.</td>
</tr>
<tr>
<td>tmcmd</td>
<td>Forces a change of state of a running transaction.</td>
</tr>
<tr>
<td>tmstat</td>
<td>Displays the status of current transactions and locks.</td>
</tr>
<tr>
<td>wlocalhost</td>
<td>Sets the name of the local host in the Windows registry. (Windows nodes only)</td>
</tr>
<tr>
<td>wlocktmr</td>
<td>Places the current Tivoli region in maintenance mode.</td>
</tr>
<tr>
<td>wmailhost</td>
<td>Specifies the mail server used Windows managed nodes.</td>
</tr>
</tbody>
</table>

### Managed node commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wclient</td>
<td>Creates a managed node.</td>
</tr>
<tr>
<td>wdate</td>
<td>Prints out the current date and time of the managed node.</td>
</tr>
<tr>
<td>wdiskspace</td>
<td>Prints the number of free kilobytes available in the specified directory (file system) of the specified managed node.</td>
</tr>
<tr>
<td>whohostid</td>
<td>Prints the host ID of the specified managed node.</td>
</tr>
<tr>
<td>wifconfig</td>
<td>Queries or changes the IP interfaces on a managed node.</td>
</tr>
<tr>
<td>winstdir</td>
<td>Prints out the path of the installation directory of the specified managed node.</td>
</tr>
<tr>
<td>winterp</td>
<td>Prints the interpreter type of the specified managed node.</td>
</tr>
<tr>
<td>wmnnode</td>
<td>Returns the properties of a managed node.</td>
</tr>
<tr>
<td>wmemsize</td>
<td>Reports the amount of physical memory of a managed node.</td>
</tr>
<tr>
<td>wrmnode</td>
<td>Removes a managed node from a Tivoli environment.</td>
</tr>
<tr>
<td>wstandalone</td>
<td>Installs a managed node or endpoint without using rsh or rexec.</td>
</tr>
<tr>
<td>wtimezone</td>
<td>Prints the time zone value of the specified system.</td>
</tr>
<tr>
<td>wuname</td>
<td>Lists operating system information.</td>
</tr>
<tr>
<td>wunstmn</td>
<td>Removes Tivoli Management Framework files from a managed node.</td>
</tr>
<tr>
<td>wxterm</td>
<td>Starts an Xterminal session on a UNIX managed node.</td>
</tr>
</tbody>
</table>

### Multiplexed distribution commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcfd</td>
<td>Provides configuration information to the endpoint daemon (lcfd), such as enabling wake-on-LAN functionality on the endpoint.</td>
</tr>
<tr>
<td>wadminep</td>
<td>Performs administrative operations on an endpoint, such as upgrading the endpoint daemon or generating the endpoint wake-up packet for wake-on-LAN operations.</td>
</tr>
<tr>
<td>Command</td>
<td>Purpose</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wdepot</td>
<td>Manages MDist 2 repeater depots.</td>
</tr>
<tr>
<td>wep</td>
<td>Performs actions on endpoint information contained in the endpoint list, such as setting a Windows endpoint to receive and control MDist 2 distributions through the use of the Tivoli Mobile Computing console.</td>
</tr>
<tr>
<td>wmdist</td>
<td>Configures MDist 2 repeaters and manages distributions.</td>
</tr>
<tr>
<td>wmdistgui</td>
<td>Starts the Distribution Status console from the managed node on which this command is run.</td>
</tr>
<tr>
<td>wrpt</td>
<td>Creates a repeater on a managed node (for both MDist and MDist 2), configures MDist repeaters, and manages MDist distributions.</td>
</tr>
</tbody>
</table>

**Notification commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wbroadcast</td>
<td>Broadcasts a message to all Tivoli desktops.</td>
</tr>
<tr>
<td>wexnpnotif</td>
<td>Expires notices from a notice group.</td>
</tr>
<tr>
<td>wslnnotif</td>
<td>Lists notices on an administrator bulletin board.</td>
</tr>
<tr>
<td>wsndnotif</td>
<td>Translates standard input into a message structure and sends it to the notification server.</td>
</tr>
<tr>
<td>wtailnotif</td>
<td>Connects to the notification server and displays new notices as they are posted.</td>
</tr>
</tbody>
</table>

**Policy commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wchkpol</td>
<td>Checks policy region members against policy.</td>
</tr>
<tr>
<td>wcrtpol</td>
<td>Creates a new policy object for a class.</td>
</tr>
<tr>
<td>wcrtpr</td>
<td>Creates a policy region.</td>
</tr>
<tr>
<td>wdelpol</td>
<td>Deletes a default policy object.</td>
</tr>
<tr>
<td>wdelpol</td>
<td>Deletes a policy region.</td>
</tr>
<tr>
<td>wgetdfpol</td>
<td>Lists a default policy object.</td>
</tr>
<tr>
<td>wgeteppol</td>
<td>Lists the body and constant values of an endpoint policy script.</td>
</tr>
<tr>
<td>wgetpolm</td>
<td>Lists the body or constant value of a default or validation policy method.</td>
</tr>
<tr>
<td>wgetpr</td>
<td>Lists the properties of a policy region.</td>
</tr>
<tr>
<td>wlspol</td>
<td>Lists available policy default and validation objects for a Tivoli resource.</td>
</tr>
<tr>
<td>wlspolm</td>
<td>Lists policy methods for a Tivoli resource.</td>
</tr>
<tr>
<td>wputeppol</td>
<td>Replaces an endpoint policy script that has been modified.</td>
</tr>
<tr>
<td>wpupolm</td>
<td>Replaces the body of a policy method.</td>
</tr>
<tr>
<td>wsetdfpol</td>
<td>Sets the default policy for a class.</td>
</tr>
<tr>
<td>Command</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wsetpr</td>
<td>Assigns the policy used in a policy region, enables or disables policy validation, and adds or removes a managed resource in a policy region.</td>
</tr>
</tbody>
</table>

**Query commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcrtqlib</td>
<td>Creates a query library.</td>
</tr>
<tr>
<td>wcrquery</td>
<td>Creates a query.</td>
</tr>
<tr>
<td>wgetquery</td>
<td>Lists information about a query.</td>
</tr>
<tr>
<td>wruninvquery</td>
<td>Queries the database for inventory information and returns a list of object IDs and object labels that match the query criteria.</td>
</tr>
<tr>
<td>wrunquery</td>
<td>Runs a query and returns the results to either standard output or a file.</td>
</tr>
<tr>
<td>wsetquery</td>
<td>Edits the properties of a query.</td>
</tr>
</tbody>
</table>

**RDBMS interface module (RIM) commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcrttrim</td>
<td>Creates a RIM object.</td>
</tr>
<tr>
<td>wgettrim</td>
<td>Lists information about a RIM object.</td>
</tr>
<tr>
<td>wmvtrim</td>
<td>Moves a RIM object to another managed node.</td>
</tr>
<tr>
<td>wrimtest</td>
<td>Verifies the connectivity and functionality of a RIM object.</td>
</tr>
<tr>
<td>wrimtrace</td>
<td>Enables or disables tracing for RIM objects.</td>
</tr>
<tr>
<td>wsetrim</td>
<td>Changes the database information for a RIM object.</td>
</tr>
<tr>
<td>wsetrimpw</td>
<td>Sets the RIM password for a RIM object database.</td>
</tr>
</tbody>
</table>

**Revision control system (RCS) commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wccl</td>
<td>Checks in RCS revisions.</td>
</tr>
<tr>
<td>wccl</td>
<td>Checks out RCS revisions.</td>
</tr>
<tr>
<td>wident</td>
<td>Identifies files.</td>
</tr>
<tr>
<td>wrcl</td>
<td>Changes RCS file attributes.</td>
</tr>
<tr>
<td>wrcsdiff</td>
<td>Compares RCS revisions.</td>
</tr>
<tr>
<td>wrcsmerge</td>
<td>Merges RCS revisions.</td>
</tr>
<tr>
<td>wrlog</td>
<td>Prints log messages and other information about RCS files.</td>
</tr>
</tbody>
</table>

**Scheduler commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdelsched</td>
<td>Removes jobs from the scheduler.</td>
</tr>
</tbody>
</table>
### Task library commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcrtjob</td>
<td>Creates a job in a task library.</td>
</tr>
<tr>
<td>wcrttask</td>
<td>Creates a task in a task library.</td>
</tr>
<tr>
<td>wcrttlib</td>
<td>Creates a task library.</td>
</tr>
<tr>
<td>wdeljob</td>
<td>Deletes a job from a task library.</td>
</tr>
<tr>
<td>wdeltask</td>
<td>Deletes a task from a task library.</td>
</tr>
<tr>
<td>wdisttask</td>
<td>Controls the distribution of task binaries for a task library.</td>
</tr>
<tr>
<td>wgetjob</td>
<td>Lists the properties of a job.</td>
</tr>
<tr>
<td>wgettask</td>
<td>Lists the properties of a task.</td>
</tr>
<tr>
<td>wlistlib</td>
<td>Lists the properties of a task library.</td>
</tr>
<tr>
<td>wrunjob</td>
<td>Runs a job in a task library.</td>
</tr>
<tr>
<td>wruntask</td>
<td>Runs a task in a task library.</td>
</tr>
<tr>
<td>wsetjob</td>
<td>Sets the properties of a job.</td>
</tr>
<tr>
<td>wsettask</td>
<td>Sets the properties of a task.</td>
</tr>
<tr>
<td>wtaskabort</td>
<td>Aborts a task transaction and rolls back any uncommitted changes.</td>
</tr>
<tr>
<td>wtll</td>
<td>Imports and exports task library definitions.</td>
</tr>
</tbody>
</table>

### Miscellaneous commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>Starts the Tivoli desktop.</td>
</tr>
<tr>
<td>vdisp</td>
<td>Checks status of all managed nodes.</td>
</tr>
<tr>
<td>waddinicon</td>
<td>Adds an icon to a Windows Program Manager group. (Windows only)</td>
</tr>
<tr>
<td>wbindmsg</td>
<td>Retrieves a translated string from a local message catalog and binds any variables.</td>
</tr>
<tr>
<td>wbkupdb</td>
<td>Backs up and restores Tivoli databases.</td>
</tr>
<tr>
<td>wcatcher</td>
<td>Saves custom dialogs in Tivoli Management Framework or a Tivoli application before an upgrade to a new version of Tivoli Management Framework or the application.</td>
</tr>
<tr>
<td>wcd</td>
<td>Changes the current working collection.</td>
</tr>
<tr>
<td>wchdep</td>
<td>Associates a dependency set with a method header.</td>
</tr>
<tr>
<td>wchkdb</td>
<td>Verifies and repairs the Tivoli database.</td>
</tr>
<tr>
<td>Command</td>
<td>Purpose</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wchknode</td>
<td>Verifies and updates references to a specific dispatcher number from parts of the Tivoli database.</td>
</tr>
<tr>
<td>wdel</td>
<td>Deletes objects from the Tivoli database.</td>
</tr>
<tr>
<td>wdtmsg</td>
<td>Sets the message that is displayed when the Tivoli desktop is started.</td>
</tr>
<tr>
<td>wdepset</td>
<td>Specifies dependencies that a method needs to run.</td>
</tr>
<tr>
<td>wgetallinst</td>
<td>Displays all instances of a resource type.</td>
</tr>
<tr>
<td>wiconv</td>
<td>Converts the characters or sequences of characters in a file from one code set to another code set.</td>
</tr>
<tr>
<td>winstendpt</td>
<td>Installs behavior for an endpoint resource type.</td>
</tr>
<tr>
<td>wlcftap</td>
<td>Sets the properties of the Tivoli Authentication Package on a Windows client.</td>
</tr>
<tr>
<td>wln</td>
<td>Links an object into a collection.</td>
</tr>
<tr>
<td>wlocpath</td>
<td>Returns the path for the localized file or directory.</td>
</tr>
<tr>
<td>wls</td>
<td>Lists the member objects of a collection.</td>
</tr>
<tr>
<td>wlsinst</td>
<td>Lists the products and patches installed in a Tivoli region.</td>
</tr>
<tr>
<td>wmerge</td>
<td>Performs a three-way file merge.</td>
</tr>
<tr>
<td>wmergeaef</td>
<td>Merge custom dialogs into Tivoli Management Framework or a Tivoli application after upgrading.</td>
</tr>
<tr>
<td>wmv</td>
<td>Moves objects between collections.</td>
</tr>
<tr>
<td>wping</td>
<td>Attempts to contact the object dispatcher on a host.</td>
</tr>
<tr>
<td>wpwd</td>
<td>Prints the current working collection.</td>
</tr>
<tr>
<td>wrefresh</td>
<td>Refreshes a Tivoli collection window.</td>
</tr>
<tr>
<td>wrm</td>
<td>Removes objects from a collection.</td>
</tr>
<tr>
<td>wrmnode</td>
<td>Removes a managed node from a Tivoli environment.</td>
</tr>
<tr>
<td>wrunase</td>
<td>Retrieves passwords and launches commands.</td>
</tr>
<tr>
<td>wsetpkey</td>
<td>Encrypts and stores passwords.</td>
</tr>
<tr>
<td>wsupport</td>
<td>Collects problem information from users to send to a customer support representative.</td>
</tr>
<tr>
<td>wtemp</td>
<td>Displays the name of the directory in which Tivoli products create temporary files.</td>
</tr>
<tr>
<td>wtrace</td>
<td>Provides information to debug methods.</td>
</tr>
</tbody>
</table>
Chapter 3. Commands alphabetically
idlarg

Extracts individual options from an option list returned by the idlimput command.

Syntax

idlarg element_offset [option_list]

Description

The idlarg command extracts individual options from an option list returned by the idlimput command. This command can also be used to get the members of a constructed type. This command returns an exit code of 0 if successful. If an error occurs, this command exits with a nonzero code.

Options

- **element_offset**
  Specifies the offset to the element in the option list that should be extracted. The first element is at offset 1, the second element is at offset 2, and so on. The value specified must be a positive integer.

- **option_list**
  Specifies a list of options in clear text format. If omitted, the command reads from standard input until an EOF is encountered.

Examples

The following example extracts the in and inout options:

```plaintext
interface test {
  exception ex { long code; string reason; };
  struct s { long l; char c; };
  s op(in s a, inout s b, out s c) raises (ex);
  attribute long attr;
};

#!/bin/sh
# shell implementation of test::op
# Get the input/inout options
inargs=`idlinput`
# $inargs may look like: "{1 'z'} {2 'w'}".
# Now separate the in and inout options.
arg_a=`idlarg 1 $inargs`
arg_b=`idlarg 2 $inargs`
# We can get to the fields of $arg_a as follows
arg_a_l=`idlarg 1 $arg_a`
arg_a_c=`idlarg 2 $arg_a`
# This will set $arg_a_l to 1 and $arg_a_c to
# 'z' respectively.
```

See Also

- idlattr
- idlcalt
- idlexception
- idlimput
- idlresult
idlatex

Gets or sets implementation attributes.

Syntax

```
idlattr -t [-a] [-v] target_object attribute type_name
idlattr -t [-g] [-v] target_object attribute type_name
idlattr -t [-s] [-v] target_object attribute type_name [value]
```

Description

The `idlattr` command gets or sets implementation (object) attributes. The `-t` option indicates that the option list contains the attribute type name; this option is required. The `-s` and `-g` options indicate a set or get operation, respectively. If neither the `-s` nor `-g` option is specified, the default is a set operation. The `attribute` option specifies the unscoped attribute name; this is the same unscoped attribute name as in the corresponding implementation construct. The `type_name` option specifies the fully scoped attribute type. If a set operation is being performed, the `value` option is the cleartext value to which the attribute should be set. If this option is omitted, the cleartext value is read from standard input.

Options

- `-t` Indicates that the option list contains the type name of the attribute. This option is required.
- `-a` Adds an attribute to the target object.
- `-g` Specifies a get operation.
- `-s` Specifies a set operation. This is the default.
- `-v` Specifies verbose mode. In verbose mode, exceptions are bound to messages and printed to standard output. The default is to write exceptions to standard output in clear text format.

```
target_object
Specifies the target object for the operation. The target object should be specified in string format.

attribute
Specifies the unscoped attribute name. This is the unscoped attribute name that is in the corresponding implementation construct.

type_name
Specifies the fully scoped type of the attribute.

value
Specifies the cleartext value the attribute is to be set to. This option is valid only for a set operation (`-s`).
```

Authorization

To get the value for an attribute: `user`, `admin`, `senior`, or `super`.

To set the value for an attribute or to add an attribute: `senior` or `super`. 
Examples

The following example shows the accessing of implementation attributes:

```c
interface test {
    exception ex { long code; string reason; };
    struct s { long l; char c; };
    s op(in s a, inout s b, out s c) raises (ex);
    attribute long attr;
};
```

Assume the following implementation of an interface test:

```c
implementation class imp_test honors test {
    struct t { long l; };
    attribute s attr1; // refers to test::s
    attribute t attr2;
    attribute unsigned long attr3; // define methods here
};
```

Assume 2001.1.15 is the object reference string to an instance of the imp_test implementation object. Its physical attributes can be accessed or modified as follows:

```
idlattr -t -g 2001.1.15 attr1 test::s
// may print '{1 'z'}
idlattr -t -s 2001.1.15 attr2 imp_test::t '{20}'
idlattr -t 2001.1.15 attr3 ulong 10
```

See Also

[idlcall]
idlcall

Provides a method of invoking Interface Definition Language (IDL) operations from the shell command line.

Syntax

idlcall [-T transaction_type] [-v] target_object operation_id [options]

Description

The idlcall command invokes IDL operations from the shell command line. The transaction_type option can be used to specify a top-level transaction, a subtransaction, a revocable transaction, or no transaction. The target_object option specifies the cleartext string representation of the target object reference. The operation_id option specifies the operation name. The operation name can be specified as the fully scoped name separated by a double colon (Common Object Request Broker Architecture [CORBA] RepositoryId) or the operation name (as in the IDL description). options specifies any input or inout options. These options are listed in the same order as in the IDL description. This command writes the inout or output options and the result, if any, to standard output in cleartext format. If the idlcall invocation results in an exception, the exception is written to standard output. If an operation requires input or inout options but none are specified on the command line, this command reads input from standard input until an EOF is encountered.

Options

- `-T transaction_type`
  Specifies a transaction type. This option can be one of the following:
  
  - `none`  No transaction
  - `revoke`  Revocable transaction
  - `sub`  Subtransaction
  - `top`  Top-level transaction

- `-v`  Specifies verbose mode. In verbose mode, exceptions are bound to messages and printed to standard output. The default is to write exceptions to standard output in clear text format.

- `target_object`
  Specifies the cleartext string format of the object reference for the target object.

- `operation_id`
  Specifies the operation name. This can be the double-colon separated fully scoped name (CORBA RepositoryId) or the operation name (as in the IDL description). The first form is more efficient because it makes fewer remote calls, but the second form is simpler to use. For IDL attributes, the operation name is the attribute name prefix with _get_ or _set_ for a get or set operation, respectively.

- `options`  Specifies the input or inout options. If input or inout options are required but not specified on the command line, the idlcall command reads input from standard input until an EOF is encountered.
Diagnostics

The `idlcall` command exits with a nonzero status if the invocation results in an exception (either in dispatching the call or raised by the method implementation). Otherwise, this command exits with 0. The exit status can be used to interpret the cleartext output.

Examples

```c
interface test {
    exception ex { long code; string reason; };
    struct s { long l; char c; };
    s op(in s a, inout s b, out s c) raises (ex);
    attribute long attr;
};
```

Assume 2001.1.15 is the clear text string representation of the object reference to an object that supports interface test. Its operations can be invoked as follows:

```
idlcall –T top 2001.1.15 test::op "{1'z'}" "{2 'w'}"
idlcall –T top 2001.1.15 op <<!EOF
{1 'z'} {2 'w'}
!EOF
idlcall 2001.1.15 _get_attr
idlcall –T top 2001.1.15 _set_attr 20
```

See Also

`idlattr`
idlexception

Raises exceptions for a shell method.

Syntax

idlexception exception_type scoped_exception_name

idlexception exception_type scoped_exception_name exception_data

Description

The idlexception command is used to raise exceptions. The script must exit with a nonzero exit code if the script raises an exception. An exit code of 0 indicates a normal return. Do not use idlexception to raise an exception from a Tivoli Extended Interface Definition Language (EIDL) command method. The Tivoli runtime will implicitly raise an exception from a command method if the command was terminated by a signal. A command method never raises an exception explicitly. Thus, a shell method refers to the EIDL shell binding only.

Options

exception_type

Specifies the exception type. This can be USER_EXCEPTION or SYSTEM_EXCEPTION.

scoped_exception_name

Specifies the fully qualified IDL exception name.

exception_data

Indicates the clear text representation of the exception structure fields. If the exception is empty, nothing is contained inside the inner parentheses. If this method is invoked without any command line options, the clear text exception is read from standard input.

Examples

interface test {
    exception ex { long code; string reason; };
    struct s { long l; char c; };
    s op(in s a, inout s b, out s c) raises (ex);
    attribute long attr;
};

#!/bin/sh
# shell implementation of test::op
# In doing some work, say we failed, and now
# want to raise the test::ex exception.
excep=`idlexception '{USER_EXCEPTION test::ex \"failed\" 99}''
# the exception must be written to stdout.
echo $excep
# must exit with a nonzero status
exit 1

To raise a system exception, enter the following:

# Raise a standard exception (also let idlexception read
# from stdin)
excep=`idlexception <<!EOF
{SYSTEM_EXCEPTION StExcep::BAD_PARAM {999 NO }}
!EOF`
idlexception

`EOF

  echo $excp
  exit 1`

See Also

idlattr  idlcall  idlinput  idlresult
idlinput

Gets the in or inout options list to a shell method.

Syntax

idlinput

Description

The idlinput command gets the in or inout options list (in clear text format) to a shell method. The options are in the same order as in the Interface Definition Language (IDL) signature. Do not use this command with Tivoli Extended Interface Definition Language (EIDL) command-style methods. A command method gets its input from the options list or standard input. Thus, a shell method refers to the EIDL shell binding only.

Examples

The following example accesses the inout options of the EIDL shell:

```idl
interface test {
    exception ex { long code; string reason; };
    struct s { long l; char c; }
    s op(in s a, inout s b, out s c) raises (ex);
    attribute long attr;
};
```

```sh
#!/bin/sh
# shell implementation of test::op
# Get the input/inout options
inargs="idlinput"
# $inargs may look like: "{1 'z'} {2 'w'}",
# the first and the second pair of curly braces
# contain the in option (a) and the inout option
# (b) respectively. The method can now access each
# individual option or their fields using idlarg.
# rest of the method goes here.
```

See Also

| idlattr | idcall | idlexception | idlresult |
idlresult

Formats the output options of a shell method.

Syntax

idlresult [options]

Description

The idlresult command formats inout or out options or any result that is returned. The inout and out options must be in clear text format and must be passed to this command in the same order that they appear in the Interface Definition Language (IDL) signature. The result, if any, follows. The options to be formatted can be specified as command line options or can be read from standard input until an EOF is encountered. Do not use this command with Tivoli Extended Interface Definition Language (EIDL) command-style methods. A command method writes its output to standard output or standard error and no formatting is necessary. Thus, a shell method refers to the EIDL shell binding only.

Examples

The following example formats the inout options of the EIDL shell:

```sh
interface test {
    exception ex { long code; string reason; };
    struct s { long l; char c; };
    s op(in s a, inout s b, out s c) raises (ex);
    attribute long attr;
};

#!/bin/sh
# shell implementation of test::op
# do some work
# return some hard coded values.
b="{1 'a'}"
c="{2 'b'}"
retval="{3 'c'}"
# the order of options is inout(b), out(c) and
# the return result. We could have also said:
#
# all=`idlresult <<!EOF
# $b
# $c
# $retval
# !EOF
#`
# all=`idlresult $b $c $retval`
# The results must be written to stdout.
echo $all
# A 0 exit code means a successful return from
# a EIDL shell method.
exit 0
```

See Also

idlattr idlcall idlexception idlinput
lcfd

Provides configuration information to the endpoint service (lcfd), including login options, port numbers, and debugging information.

Syntax


Description

The lcfd command provides configuration information to the endpoint service, including login options, port numbers, and debugging information. It can also start the endpoint service installed on an endpoint. UNIX endpoints can use the lcfd.sh command to start and stop the endpoint. You can use the -i option to install the the endpoint service as a Windows operating system service. The -r option removes an existing service from the Windows Service Manager. Internetwork Packet Exchange (IPX) can be used to connect to endpoints only on NetWare and Windows operating systems.

The -S options can be used with the -L option of the winstlcf command to allow service dependencies to be created for the endpoint service when it is installed. However, when using the -S option with the -L option, the service_name option, if present, is ignored because the name of the service being installed is used. The -S option can also be used after the endpoint is installed to define a new service dependency.

The -S option can be used with the -i option to create service dependencies during the installation of a new endpoint service. However, this results in the creation of two endpoint services: one without the dependency and one with the dependency. In addition, when using the -i option, service_name is ignored because the name of the service being created is used.

The -S option can also be used to remove dependencies for a previously installed endpoint. Because the -S option cannot be used without suboptions, service_name or an empty string (""") is required to remove all dependencies from an existing endpoint service.

Options

-b library_dir

Specifies the path to the configuration library, which contains the shared libraries required by an endpoint. This option does not apply to NetWare.

-C directory_name

Specifies the name of the endpoint’s current working directory. This directory contains configuration files needed for startup and the method cache.

-d level

Defines the level of debug messages written to the lcfd.log file. The default value is 1. The following are valid entries:

0 No message logging
Minimal logging (default)
Tracing and moderate output
Detailed information and tight loops
Data

Note: Level 1 is the default value. Level 4 message logging generates a large number of messages. Level 2 or 3 is recommended for troubleshooting.

-D option=value
Reconfigures the endpoint during startup using one or more of the following options. Configuration information is stored in the last.cfg file on the endpoint. Some of these options can also be set with command line options.

? Displays the usage statement for this command.

address_notif_interval=seconds
For Dynamic Host Configuration Protocol (DHCP) environments, specifies an endpoint timeout interval to wake up from its idle state and attempt to notify the gateway of its current IP address. This option is only for endpoints that might change IP addresses without the endpoint daemon restarting. The recommended value for this option is 300 seconds. The default value for this option is 0, meaning that the endpoint daemon does not notify the gateway.

address_notif_retry_count=count
Specifies the number of times that the endpoint will retry sending its IP address to the gateway, each address_notif_interval seconds, before attempting to contact another gateway in its login interfaces list. The default is 0.

allow_proxy_upcalls=true | false
Specifies whether all communication between the endpoint and its assigned gateway use the configured target port on the endpoint. If set to true, the endpoint and gateway communicate using the target port only. Before endpoint can use proxy upcalls, the endpoint must establish a connection to a proxy-capable gateway. If set to false, the endpoint and its assigned gateway communicate as usual. This option modifies the value of allow_proxy_upcalls in the last.cfg file. The default is false.

bcast_disable=1 | 0
Disables broadcast (1). When you disable broadcasting, you must use the lcs.login_interfaces option to define preferred gateways. If the endpoint protocol uses IPX, the extended broadcast uses the Routing Information Protocol (RIP) protocol and is able to send login packets within a 5-hops radius. The default is 0.

cache_limit=max_size
Specifies the maximum size of the method cache. After this maximum size is reached, the least recently used methods are deleted to make room for current methods.

cache_loc=cache
Identifies or changes the name of the method cache created and
maintained on the endpoint. This option can also be used to change the location of the method cache when submitted with a full path name.

config_path=full_path
Identifies the absolute path to the last.cfg configuration file. Editing this option is not recommended.

debug_flags=debug_level
Enables the user to attach a debug tool to a running method. Editing this option is not recommended.

depot_dir=path
Specifies the directory on the endpoint where multicast distributions are stored until the distributions are installed. This option modifies the value of depot_dir in the last.cfg file. The default directory is $LCF_DATDIR/depot. If a relative path is specified, the depot directory is relative to the value of the $LCF_DATDIR variable.

detect_address_change=TRUE | FALSE
Specifies whether the endpoint detects changes to its network interface configuration and, if necessary, takes corrective action. When this option is set to TRUE, the endpoint monitors its network interface configuration for address changes. If the listening address for the endpoint changes, the endpoint attempts to log back in to its gateway. (The listening address is set using the local_ip_interface option.) This option applies to Windows 2000, Windows XP, and Windows Server 2003 systems only. The default is FALSE.

diag_interval=seconds
Specifies the interval, in seconds, that an endpoint waits between running a series of self-diagnostic tests. These tests help you determine the ability of an endpoint to run tasks and methods. The tests include checking temporary file space and permissions and cache space and permissions. On Windows operating systems, the tests also check token creation and process spawning. After the tests complete, the results are stored in the lcfd.log file. The results are sent to the current gateway if the endpoint is logged in. If this value is set to zero, a test will only be performed when requested by a gateway. The default is zero. The acceptable range of values is 900 seconds (15 minutes) to 86400 seconds (1 day).

diag_temp_space=bytes
Sets the minimum desired temporary space, in bytes, for running the self-diagnostic tests controlled by the diag_interval option. If this value is set to zero, the tests are not run. The default is zero. The minimum value is 1024.

fail_if_pref_port_busy=1 | 0
Specifies whether the endpoint should not start if its preferred endpoint port is inaccessible. If set to 1, the endpoint does not start when the port is unavailable. If set to 0, the endpoint will try an alternate port when the preferred port is unavailable. This option modifies the value of fail_if_pref_port_busy in the last.cfg file. The default is 0.

filefree_upcalls=TRUE | FALSE
Specifies whether consolidated upcalls write the upcall data to
lcfd

disk. This setting is valid only when allow_proxy_upcalls is set to TRUE. In this case, the default is to write the upcall data to disk. However, if you do not want upcall data written to disk, you can set filefree_upcalls to TRUE and all upcall data is transferred through the wire. This option modifies the value of filefree_upcalls in the last.cfg file. The default is FALSE.

gateway_port=port_number
Identifies the port on which the gateway monitors endpoint communications. The default value is 9494. This option can also be set using the -P option.

http_disable=value
Specifies the level of functionality for the Web browser accessible on the endpoint.

0 Anyone can use a browser to reconfigure the endpoint, This is the default value.

1 Anyone can use a browser to view the configuration data, but no one can use a browser to reconfigure the endpoint.

2 No one can use a browser to view or reconfigure the endpoint.

3 Everyone must provide authentication (user ID and password) to view or reconfigure the endpoint.

httpd_pw=password
Identifies the password that prevents security intrusion to the endpoint configuration file. This password is stored by the endpoint manager. To change this password, use the wep command.

interp=interpreter_type
Identifies the interpreter type of the endpoint. Editing this option is not recommended.

lcfd_alternate_port=port_number
Identifies an alternate port on which the endpoint service monitors gateway communications if endpoint service cannot contact its default port (set with the -P option) during startup. The default value is 9496.

lcfd_autostart=TRUE | FALSE
Specifies whether Windows endpoints set the endpoint service to start automatically. When this value is set to TRUE, a Windows endpoint checks the Windows endpoint service once during endpoint startup and once during endpoint shutdown. If the endpoint service is set to Manual, the endpoint resets the service to Automatic. When this value is set to FALSE, the endpoint does not check the setting of the Windows endpoint service. The default value is FALSE.

lcfd_port=port_number
Identifies the port on which the endpoint service monitors gateway communications. The default value is 9495. This option can also be set using the -P option.

lcsgateway_address=IP_address | IPX_address
Changes the login gateway after the endpoint has successfully logged in. If the gateway has not previously logged in, use the
lcs.login_interfaces option to provide one or more gateways through which the endpoint can log in. For NetWare and Windows endpoints using IPX, to log in to a gateway located outside a 5-hops radius, you must specify the IPX address (not host name).

lcs.login_interfaces=address[+port][;address[+port]]...
Specifies the IP address or host name (or IPX address or server name) and port number of one or more gateways to which an endpoint sends its login packet. This option is required for the endpoint to log in to a gateway on a different subnet or to log in to a specific gateway when two or more exist on a subnet. If your gateways and endpoints are separated by a network address translation (NAT) device, specify host names instead of IP addresses. Multiple addresses must be separated by colons. You can also use the -g option to list one or more gateways.

Note: This option does not specify the gateway to which an endpoint is ultimately assigned. The endpoint manager determines gateway selection and assignment. If the endpoint has successfully logged in to a gateway, use the lcs.gateway_address option to change the gateway.

lcs.machine_name=endpoint_label
Identifies the endpoint label as shown in wlookup or wep. You can also use the -n option to identify an endpoint label.

lcs.machine_unique_id=ID_string
Identifies the unique identifier of the endpoint. The ID_string value must be a unique string within the Tivoli environment. The length of the string must be 36 or fewer characters. This value is stored in the $LCF_DATDIR/lcf.id file.

local_ip_interface=IP_address
For endpoints with multiple IP addresses, allows connections on the specified IP address. If the local_ip_interface option is specified, the endpoint binds to the provided address instead of 0.0.0.0, and connections are only accepted on that interface.

log_threshold=integer
Specifies the level of detail written to trace files for the identified endpoint. This option modifies the value of log_threshold in the last.cfg file. The integer value specified can be in the range of 1 to 4, where 1 provides the least level of detail and 4 provides the most. The default is 1.

logfile=full_path
Identifies the absolute path to the file in which status messages are logged. The default log file is lcfd.log. Editing this option is not recommended. Use the -l option to change the name of the log file.

login_attempts=value
The number of directed login attempts on a gateway before the endpoint moves to the next gateway in the list. A directed login attempt is an attempt to log in to either the last known gateway or to a gateway in the login interfaces list. The default is 3.

login_interval=seconds
Specifies the number of seconds between initial login attempts. The default is 1800 seconds (30 minutes).
login_mode=mobile | non_mobile
For Windows endpoints only, enables a user at the specified endpoint to receive and control MDist 2 distributions through the Tivoli Mobile Computing console. Specify the non_mobile option to disable console functions on the endpoint.

Note: For this to work, a Tivoli administrator must first use the wep command with the set login_mode –s variable options.

You also can enable mobile support by using the wep command with the set login_mode –m mobile options.

login_timeout=seconds
Specifies the number of seconds that an endpoint waits for a response to a directed login attempt. A directed login attempt is an attempt to log in to either the last known gateway or to a gateway in the login interfaces list. The default value is 300 seconds (5 minutes).

log_queue_size=max_size
Specifies the maximum amount of memory (measured in bytes) used for the log queue. Only LogQ messages are sent to the log queue. If an exception occurs, the entire queue is printed to the screen. The valid range is 1024 through 102400.

log_size=max_size
Specifies the maximum size (in bytes) of the log file. The valid range is 10240 through 10240000.

log_threshold=debug_level
Identifies the level of debug logging. This option can also be set using the –d option.

preload_loc=location
Specifies the location where the preloaded methods are stored on S/390 endpoints. This option modifies the value of preload_loc in the last.cfg file.

protocol=TCPIP | IPX
Specifies the protocol on which the endpoint service monitors gateway communications. Supported protocols are as follows:

TCPIP Specifies Transmission Control Protocol/Internet Protocol (TCP/IP). This is the default. You cannot disable TCP/IP.

IPX Specifies Internetwork Packet Exchange (IPX).

To specify both TCP/IP and IPX protocols, specify the option as protocol=TCPIP,IPX. You also can set this option using the –x option. Note that you cannot turn off the TCP/IP protocol for a gateway.

recvDataNumAttempts=count
Specifies how many times to receive or attempt to receive data on the current connection before closing the connection. The default is 10.

recvDataQMaxNum=connections
Specifies how many connections to hold in the pending connection queue. A connection that is waiting for data is considered a pending connection and is added to the queue. After this limit is
reached, all additional connection attempts are rejected until a pending connection is closed. The default is 50.

recvDataTimeout=seconds
Specifies how many seconds that a new connection waits for incoming data before requeuing the connection. If the pending connection queue is full, the connection request is rejected. The default is 2.

repair_accts=TRUE | FALSE
Specifies whether the endpoint tests the installation and configuration of the tmersrvd account and the Tivoli_Admin_Privileges group and makes repairs as required. When this option is set to TRUE, the endpoint performs this check once during startup, and any corrective actions are noted in the lcfd.log file. When this option is set to FALSE, this check is not performed. The default value is FALSE.

repair_tap=TRUE | FALSE
Specifies whether the endpoint tests the installation and configuration of Tivoli Authentication Package and makes repairs as required. When this option is set to TRUE, the endpoint performs this check once during startup, and any corrective actions are noted in the lcfd.log file. When this option is set to FALSE, this check is not performed. The default value is FALSE.

run_dir=directory_name
Identifies the directory from which the endpoint daemon runs. Editing this option is not recommended.

run_timeout=seconds
This option is deprecated. Use the udp_interval option to specify the wait time (in seconds) before a communication timeout occurs following a successful login.

start_delay
Delays the startup of the endpoint service in cases where an endpoint and a gateway coexist on the same machine. If this option is not used in the above circumstances, endpoint login either takes a long time, or it might not occur at all.

start_timeout=seconds
Specifies the wait time (in seconds) before a communication timeout occurs during login. The default value is 120.

udp_attempts=number
Specifies the number of times an endpoint attempts to connect to a gateway during a broadcast login attempt. The default value is 6. After this number of login attempts is exceeded, the endpoint waits for the number of seconds indicated by the login_interval value, and then restarts the login cycle by attempting to log in to the gateway or gateways specified in the login interfaces list.

udp_interval=seconds
Specifies the number of seconds that the endpoint waits for a response from a gateway before attempting another broadcast login. The default value is 300 seconds (5 minutes).
lcfd

**upcall_retry_count=count**

Specifies the number of attempts to retry after an upcall failure. This option modifies the value of **upcall_retry_count** in the last.cfg file. The default is 0.

**web_post_interval=seconds**

Specifies the interval, in seconds, at which an endpoint posts event and state change information to a Web server. Information is sent only if an event or state change occurs within this interval. The minimum value is 300 seconds (five minutes). The maximum value is 86,400 seconds (one day). The default value is zero. When this option is set to zero, endpoint Web posting is disabled.

**web_post_url=URL**

Specifies the URL to which an endpoint posts event and state change information. Enter the syntax for **URL** as shown in the following example:

```
http://web_server_name:port/relative_path_to_cgi_script/script_name
```

**Notes:** You do not need to specify the port number if you are using the default port of 80.

**wol_enable= 0 | 1 (OS/2 and Windows operating systems only)**

Specifies whether the Network Interface Card (NIC) on the box supports Wake on Lan. This option modifies the value of **wol_enable** in the last.cfg file. The default is 0.

**-g address[+port]...**

If the endpoint uses TCP/IP, this option specifies the IP address or host name and, optionally, the port number of one or more gateways to which an endpoint sends its login packet. If the endpoint uses Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX), this option specifies the IPX address or server name and port number of one or more gateways to which an endpoint sends its login packet. The default port number is 9495. This option is required for the endpoint to log in to a gateway on a different subnet or to log in to a specific gateway when two or more exist on a subnet.

**Notes:**

- If your gateways and endpoints are separated by a network address translation (NAT) device, specify host names instead of IP addresses. Multiple addresses must be separated by colons.
- For NetWare and Windows endpoints to log in to a gateway located outside a 5-hops radius, you must specify the IPX address (not server name).

**-H**

On OS/2 only, enables you to start the endpoint service on OS/2 systems in detached mode. This option removes lcfd.exe from the task list to prevent users from inadvertently stopping the endpoint service. This option does not hide the process from other means of detection such as PSPM2, psstat, or KILLFEATUREENABLE=YES in the config.sys file.

**-i**

Installs the endpoint software as a Windows service on the specified endpoint. This option is valid on Windows operating systems only.

**-l file_name**

Specifies the name of the log file to which status and error messages are written. The default file name is lcfd.log.
-n endpoint_label
Identifies the endpoint label as shown in wlook up or wep. NetWare, Windows, and OS/2 endpoints communicating over TCP/IP use the host name of the machine as the default. NetWare endpoints using IPX use the server name as the default. Windows endpoints using IPX use the computer name as the default.

-P gateway_port
Specifies the port number on which the gateway monitors endpoint communications.

-P endpoint_port
Sets the port number on which the endpoint monitors gateway communications.

-r service_name
Stops the specified endpoint service. This option is valid for Windows operating systems only.

-s
Starts an endpoint as a console application, printing all messages to the screen and to the endpoint log file. On OS/2 and NetWare endpoints, this option is ignored.

-S [service_name:]
dependency | dependency_group
Specifies service dependencies to be created when the endpoint service is installed. To specify a service dependency, use the following options:

service_name
The name of the endpoint service.

dependency
The name of the service on which the endpoint service will depend.

dependency_group
The name of the group of services on which the endpoint service will depend.

This option can be used with the -L option of the winstlcf command.

-w 0 | 1
Enables Wake on LAN functionality on the endpoint (1). When an endpoint enabled for Wake on LAN performs a normal login, the endpoint sends the gateway the information necessary to generate the wake-up packet. By default, Wake on LAN is disabled (0). For more information about Wake on LAN, see Tivoli Management Framework Planning for Deployment Guide.

-x TCP/IP | IPX
Specifies the protocol used by the endpoint. Supported protocols are the following:

TCP/IP
Specifies Transmission Control Protocol/Internet Protocol (TCP/IP). This is the default. If you do not specify the -x option, the endpoint uses TCP/IP.

IPX
Specifies Internetwork Packet Exchange (IPX).

To specify both TCP/IP and IPX protocols, specify the option as -x=TCP/IP,IPX. You also can set this option using the protocol option. Note that you cannot turn off the TCP/IP protocol for a gateway.
Authorization

No Tivoli authorization role is required.

Examples

1. The following example provides usage information about the command:
   `lcfd -s -D?`
2. The following example starts the local endpoint using configuration information contained in the last.cfg file:
   `lcfd`
3. The following example starts the endpoint as a service on the local machine and sets the debug level to 3, which causes the endpoint to log all messages to the lcfd.log file:
   `lcfd -i -d 3`
4. The following example restarts the local Windows endpoint. The endpoint logs in to a gateway outside its subnet (with host name zeus) on port 27246. The `–p` option indicates both the endpoint and gateway use port 27246.
   `lcfd -p 27246 -g zeus+9494`
5. The following example starts the local endpoint as a service and specifies that it uses SPX/IPX as a protocol. It also specifies that the endpoint logs in to a gateway with the specified IPX address, listening to a specified port.
   `lcfd -x IPX -g 4132AF12.000000000001+41204 -i`
6. The following example starts the endpoint service on a Windows operating system and specifies that it update the `log_threshold` option to debug level 3. Note that in this situation the option is not preceded with a dash, but is preceded with a slash.
   `net start lcfd /d3`

See Also

`lcfd.sh`
lcfd.sh

Starts or stops the lcfd endpoint daemon on UNIX endpoints.

Syntax

```
lcfd.sh start [lcfd_options]
lcfd.sh stop [lcfd_options]
```

Description

The lcfd.sh command is a wrapper for the lcfd command. The lcfd.sh command contains the links to platform-specific shared libraries required on UNIX endpoints. The lcfd.sh command takes the same options as the lcfd command and resides in $LCF_DATDIR directory. These options are then passed to the lcfd command when it is invoked. See the lcfd command for a detailed explanation of the command options.

The lcfd.sh command is run locally on UNIX endpoint and invoked from the $LCF_DATDIR directory.

Options

```
start     Starts the lcfd endpoint daemon.
stop      Stops the lcfd endpoint daemon.
lcfd_options
          See the lcfd command for a detailed explanation of the command options.
```

Authorization

No Tivoli authorization is required.

Examples

The following example is run locally on a UNIX endpoint and stops the endpoint daemon on that machine:

```
lcfd.sh stop
```

See Also

```
lcfd
```
logls

Creates a readable version of a transaction log file.

Syntax

glog [-D] [-o] [-f] [-l] [-s] [-k directory] [-m max_dump_data] log_name...

Description

The logls command creates a readable version of the specified transaction log files. It is primarily a diagnostic tool for debugging the transaction manager. You might be asked by your support provider to run this command to help in debugging if a problem is encountered.

Note: Modifying transactions should not be used as a routine method of problem determination. Altering transaction data can have serious and irreversible affects on the operation of the Tivoli region. Use transaction information to find out more about what is happening in the Tivoli region and use other methods to attempt to rectify problems.

The transaction log for the object dispatcher service is the odb.log transaction log. This file is located in the database directory.

If the object dispatcher service is running, you might want to shut it down or synchronize the database. Because of currently running transaction, the log file can be incomplete.

To flush the transaction log file, use the db_sync option or the odadmin command.

Options

-D    Prints the data in the log records.
-f    Lists only the “forward” log records.
-k directory
     Specifies the directory to prepend to each log file name.
-l    Prints the log headers.
-m max_dump_data
     Specifies the maximum amount of data to dump. This option should be
     used with the -D option. If the max_dump_data option is not specified, the
     default amount is 64 bytes.
-o    Lists only the “old pages” log records.
-s    Lists only the log headers. No “forward” or “old pages” log records are
     listed.

log_name...

    Specifies the name of the log file to be processed. You can specify multiple
    log files.

Authorization

You must have read permission for the log files you want to list.
Examples

The following example creates a readable version of a specified transaction log file:

```
logls -k /var/spool/Tivoli/myhost.db odb.log
```

Database update records:
old page 0 8248
insert "0.0.0"
insert <0.0.0\x00.attr_bootcount\x00>
(0:0:0:1:0) replace <0.0.0\x00.attr_ids\x00>

Database transaction state transitions:
prepare transaction {0:0:0:1:0}
complete transaction {0:0:0:1:0}
abort transaction {202020:1,202020:1,1:75}

Database event and undo callbacks registered:
{2020201,202020:1,1:61} undo [1:0:286748945] 2000.1.3 undo_callback
{2020201,202020:1,1:61} Event-prepare 2000.1.3 prepare_callback
{2020201,202020:1,1:61} Event-complete 2000.1.3 commit_callback
{2020201,202020:1,1:61} Event-abort 2000.1.3 abort_callback

See Also

tmcmd tmstat
**objcall**

Performs an object call from the shell.

**Syntax**

```
```

**Description**

The `objcall` command requests the specified object to run the specified method with zero or more options. The standard output and standard error for the method are sent to the standard output and standard error for the `objcall` command. This command exits with the method exit code. This command is only used for non-IDL (Interface Definition Language) methods.

**Options**

- `-a` Performs the object call asynchronously.
- `-b` Passes the `objcall` command standard input to method standard input. If this option is not specified, the method gets an empty standard input.
- `-c group:role:` Performs the object call with the specified group and specified roles. The caller can only specify roles that the caller has. If omitted, the method runs with all roles for the caller.
- `-e` Passes the `objcall` command environment as the method environment. If this option is not specified, the method is given a default miniature environment.
- `-F file_descriptor` Specifies the file descriptor number to which to write status information.
- `-k length` Reads the number of bytes specified by the `length` option from standard input for the key value. If omitted, no key is used.
- `-n` Starts the method and exits the `objcall` command asynchronously without waiting for the method to return results.
- `-p port` Specifies the local object dispatcher port number.
- `-s` Creates Inter-Object Messaging (IOM) keys for sending input and output to and from a method. This option should be specified only if the method being called expects these keys as input.
- `-T transaction_type` Specifies a transaction type. This option can be one of the following:
  - `none` No transaction
  - `revoke` Revocable transaction
  - `sub` Subtransaction
  - `top` Top-level transaction
**Examples**

1. The following example invokes the `get_oserv` method against the base object Tivoli server management region server is returned:
   ```
   objcall 0.0.0 get_oserv 2248904349.1.2
   ```

2. All Tivoli regions have a unique 10-digit number that is the first part of every object’s three-part OID. The particular Tivoli region queried in example 1 had a unique identifying number of 2248904349. Instead of typing those 10 digits in any given command, you can substitute the `$TMR` variable (after you have initialized your Tivoli environment).

   In the following example, the middle part of the OID, in this case, the 1, designates which managed node contains the object referenced by the third part of the OID (the 2). The first managed node created is the Tivoli server, which always receives a middle number of 1. The third part of the OID designates some type of object belonging to the managed node. In this case, the 2 is the numerical designation for the `oserv` process itself.

   The following example invokes the `boot_method` method, with the `list` argument, against the object referred to as `$TMR.1.2`. This objcall asks the `oserv` object to list all the methods that are started automatically when the Tivoli server (dispatcher 1) is booted.
   ```
   objcall $TMR.1.2 boot_method list
   ```

3. The following command invokes the `query` method against the `oserv` process (the 2 in the OID) on the Tivoli server (the 1 in the OID), in the region which has the unique identifying number of 2248904349. The argument supplied to the `query` method is `install_dir`, which results in the database call returning the path name to the installation directory. In this case, the Tivoli server is running a UNIX operating system.
   ```
   objcall 2248904349.1.2 query install_dir
   /usr/local/Tivoli/bin
   ```

   To invoke this command using the `$TMR` variable to substitute for the unique 10-digit number, and to run the query against the fifth managed node created (the 5 in the OID), enter the following command. In this case, the managed node queried is a machine running a Windows operating system.
   ```
   objcall $TMR.5.2 query install_dir
   C:\tivoli\program files\bin
   ```

---

**Authorize**

Specifies one or more arguments for the method. If this option is not specified, the method does not get any arguments.

**method** Specifies the method to be run.

**object_id** Specifies the object ID of the object that is to run the method.

**Note:** If both the `-b` and `-k` options are specified, the key (`-k`) is read first.
objcall

See Also

idlcall
odadmin

Manages object dispatchers.

Syntax

odadmin [option [suboption]]

Description

The odadmin command provides a command line interface to many run-time configuration setting and management operations for an object dispatcher. The list of supported functions includes the following:

- Allowing or disallowing client installations
- Synchronizing object databases
- Getting or setting object dispatcher environments
- Listing information about object dispatchers
- Recycling object dispatchers
- Performing remote Tivoli region operations
- Setting encryption levels and passwords
- Setting network security
- Setting downed-host checking options
- Starting and shutting down object dispatchers
- Adding or changing platform and application license keys

It is recommended that you back up all object databases before you use the odadmin command to change the low-level configuration of a Tivoli region. There may be better methods (using the Tivoli desktop or Tivoli commands other than the odadmin command) to perform some of the operations that the odadmin command allows you to perform. If you are not explicitly familiar with the implications of the odadmin command, call your Support provider before attempting to perform any odadmin operations.

Options

allow_client_install TRUE | FALSE
Sets an installation flag to permit or disallow adding additional object dispatchers to the local region. You must have the super or senior role to run this option.

allow_dynamic_ipaddr TRUE | FALSE
Enables or disables dynamic Internet Protocol (IP) address assignment support (Dynamic Host Configuration Protocol [DHCP]) within the local Tivoli region. The default value is FALSE.

db_sync [object_dispatcher... | clients | all]
Flushes the object database state to disk. You can flush the specified object databases (object_dispatcher), all object databases on managed nodes (clients), or all object databases (all) to disk. Requires the super or senior role.

environ
Gets or sets the method environment for specific object dispatchers. The method environment is the environment variables that are set inside the process of the method when the method executes. Default environment
variables are set in addition to the user-specified variables displayed when using this option. Requires the super or senior role.

The following suboptions are available with the environ option:

get [object_dispatcher... | clients | all]
- Gets the method environment for one or more specific object dispatchers (object_dispatcher...), all object dispatchers on managed nodes (clients), or all object dispatchers (all).

set [object_dispatcher... | clients | all]
- Sets the method environment for one or more specific object dispatchers (object_dispatcher...), all object dispatchers on managed nodes (clients), or all object dispatchers (all). The method environment is read from standard input.

get_allow_NAT
- Displays the value of the set_allow_NAT option, which enables or disables network address translation (NAT) support.

get_port_range
- Gets the port range setting for Inter-Object Messaging (IOM) channel communication ports and Tivoli communication between managed nodes. To set the port range, use the set_port_range option.

get_rpc_max_threads
- Retrieves the maximum number of concurrent remote procedure call threads handled by the dispatcher. You can reset this number with the set_rpc_max_threads option.

help [suboption]
- When invoked with no option, gives top-level help about the available options for the command. If an option is specified, gives help for the specified option. If no help is available for the specified option, the top-level help menu is displayed. This option requires no specific role.

odinfo [object_dispatcher... | clients | all]
- Provides information about specific object dispatchers (object_dispatcher...), all object dispatchers on managed nodes (clients), or all object dispatchers (all) in an installation. odinfo is the default option if you invoke the odadmin command without specifying any options. If you do not specify any object dispatcher numbers or the clients or all option, the odinfo option returns information about the local object dispatcher. You must have the super, senior, admin, or user role to run this option.

The odinfo option provides the following object dispatcher information:

Copyright information
- Indicates the IBM copyright information.

Region
- Indicates the Tivoli region number, which is a unique number encoded within the license key.

Dispatcher
- Indicates the server or dispatcher number within the Tivoli region. Dispatcher number 1 indicates a Tivoli server. The Tivoli dispatcher number is based on the installation order.

Interpreter Type
- Indicates the machine interpreter type.
Database directory
   Indicates the path for the local Tivoli object database.

Install directory
   Indicates the path for the object dispatcher installation directory and the location of the binaries.

Inter-dispatcher encryption level
   Indicates the encryption type used when managed nodes pass messages between themselves.

Remote client login allowed
   Enables you to make a remote desktop connection using Tivoli Desktop for Windows.

Force socket bind to a single address
   Indicates whether to force socket bind to a single address. For example, if this statement is FALSE and you have multiple network interface cards (NICs), the oserv opens port 94 and binds to all IP addresses, one for each NIC. (TCP/IP lets the oserv listen on port 94 on all IP addresses.) If this statement is TRUE, it indicates that the oserv binds to only port 94 on one IP address.

Perform local hostname lookup for IOM connections
   Indicates that Inter-ORB Messaging (IOM) will use the IP address passed to make a connection back to the initiator of the IOM request. It will not use the host name passed to look up the IP address.

Use Single Port BDT
   Indicates whether single-port Bulk Data Transfer (BDT) is enabled (TRUE) or disabled (FALSE) for this node.

Use communication channel check
   Indicates whether a communication channel check is enabled (TRUE) or disabled (FALSE) for this node.

Communication check timeout
   Indicates the number of seconds after which a process sends a ping message to determine whether an idle IPC channel has been terminated.

Communication check response timeout
   Indicates the number of seconds after which a Tivoli process determines that an IPC channel is terminated.

Oserv connection validation timeout
   Indicates the number of seconds that the object dispatcher attempts to validate an incoming connection before rejecting the connection.

Port range
   Identifies the range of ports that the Tivoli environment is allowed to use.

Single Port BDT service port number
   Indicates the port that the BDT service uses on this node.

Network Security
   Indicates the network security level of a managed node. none specifies that Secure Sockets Layer (SSL) is not used by the managed node, except when it is SSL-capable and is communicating with a FORCE_SSL managed node. SSL
that the managed node uses SSL when communicating with other SSL managed nodes. **FORCE_SSL** specifies that the managed node only communicates using SSL.

**SSL Ciphers**
Indicates the cipher list (in order of preference) used with SSL network security. The keyword **default** indicates the default Tivoli cipher list of 05040A030609.

**ALLOW_NAT**
Indicates whether network address translation (NAT) support is enabled (TRUE) or disabled (FALSE).

**Note:** On UNIX operating systems, the **odadmin** command also displays the library path in effect for Tivoli operations. On Windows operating systems, the dynamically linked libraries (DLLs) are stored in the binary directory instead of a separate library directory.

For a Tivoli server, the following information is also provided:

**State flags in use**
Indicates whether the oserv ping cache is consulted. For more information about communicating between managed nodes, see *Tivoli Management Framework Maintenance and Troubleshooting Guide*.

**State checking in use**
Indicates whether the host state information is kept up-to-date with polling (TRUE) or collected implicitly (FALSE).

**State checking every x seconds**
Indicates the interval at which state checking is performed.

**Dynamic IP addressing allowed**
Indicates whether Dynamic Host Configuration Protocol (DHCP) support on managed nodes is enabled.

**Transaction manager will retry messages x times**
Indicates the number of inter-ORB retries for communicating with another oserv.

**odlist [suboption]**
Lists or edits information about the dispatchers in an installation. You must have the super or senior role to run this option. Note that the **odlist** information is cached and the Flags column output might not accurately reflect the most recent connection or communication state of the managed node. To force a new polling of the state, invoke a command that actually contacts the managed node in question (such as the **wping** command), and then reinvoke the **odadmin** command with the **odlist** option. If you specify the **odlist** option without any suboptions, the following information is provided in a columnar format about each currently attached object dispatcher:

**Disp** The object dispatcher number.

**Flags x/y/z**
There are three flags. If the first flag (x) is c, it indicates that the object dispatcher is connected to the remote dispatcher listed in the display.

If the first flag is ?, it indicates that the cached state is out-of-date, and the state of the connection to the remote dispatcher is
unknown. The state is updated only in certain circumstances. (See Tivoli Management Framework Maintenance and Troubleshooting Guide for details.)

If the first flag is -, it indicates that the remote dispatcher is down.

The second flag (y) is always t, which indicates that the object dispatcher is trusted.

The third flag (z) is always -, which indicates that the dispatcher is not a special dispatcher.

Hostname(s)
The host name and aliases of the client on which the object dispatcher is located.

IPaddr
The IP addresses of the client on which the object dispatcher is located.

Port
The listening port for the object dispatcher.

Region
The number identifying the installation in which the object dispatcher is located.

The following suboptions are available with the odlist option:

list_od
Lists members of the region.

add_ip_alias object_dispatcher {IP_address \ host_name}
Adds an IP address alias for an object dispatcher. The request fails if the new IP address and object dispatcher port number match another object dispatcher IP address and port number. This suboption must be invoked from the Tivoli server.

delete_ip_alias object_dispatcher IP_address
Deletes an IP address alias for an object dispatcher. This suboption must be invoked from the Tivoli server.

add_hostname_alias object_dispatcher IP_address host_name
Adds a host name alias to an IP address associated with an object dispatcher. This suboption must be invoked from the Tivoli server.

delete_hostname_alias object_dispatcher IP_address host_name
Deletes a host name alias for an IP address associated with an object dispatcher. This suboption must be invoked from the Tivoli server.

change_ip object_dispatcher IP_address [TRUE \ FALSE]
Changes the primary IP address associated with an object dispatcher. The request fails if the new IP address and object dispatcher port number match another object dispatcher IP address and port number. This suboption must be invoked from the Tivoli server.

rm_od object_dispatcher
Removes an object dispatcher from the installation. The specified object dispatcher must be shut down before it is removed. A list of object IDs for objects that the dispatcher owned is displayed. The objects are removed, but references to the objects remain. This suboption must be invoked from the Tivoli server.
The `rm_od` option should only be run at the direction of your Support provider when a client installation has failed and it is impossible to recover. The usual method of deleting a client is with the `wrmnode` command.

**objects object_dispatcher**
Displays a list of the object IDs of objects owned by the object dispatcher.

**reexec [object_dispatcher... | clients | all]**
Recycles (stops and restarts) the local object dispatcher. You can recycle specified object dispatchers (*object_dispatcher*), all object dispatchers on managed nodes (*clients*), or all object dispatchers (*all*). The Tivoli server object dispatcher cannot be recycled from another system. Requires the `super` or `senior` role.

**region [suboption]**
Provides remote Tivoli region operations. You must have the `super` or `senior` role to run this option. The following suboptions are available, and should only be run at the direction of your Support provider to resolve low-level problems that cannot be handled from the Tivoli desktop or by the interregion commands.

**add_alias region IP_address [host_name...]**
Adds a host name or IP alias for a remote Tivoli server.

**add_group_map region remote_group local_group**
Adds an object group map between Tivoli regions.

**add_group_id_map region remote_name local_name**
Adds mapping from user group names in the remote Tivoli region to group names in the local Tivoli region.

**add_region region host port [crypt]**
Integrates an additional region. The password is read from standard input. You are prompted for the password if you run this command from a terminal device (tty).

**add_role_map region remote_group remote_role local_role**
Adds a role map within a group map.

**add_user_id_map region remote_name local_name**
Adds mapping from user login names in the remote Tivoli region to login names in the local Tivoli region.

**change_region region host port [crypt]**
Changes configuration information for a remote Tivoli region. The password is read from standard input. You are prompted for the password if you run this command from a terminal device (tty).

**Note:** If you do not specify the region password, the region password is deleted. When the region password is deleted, the odlist.dat file is corrupted for the changed region.

**delete_alias region IP_address [name...]**
Deletes a host name or IP alias for a remote Tivoli server.

**delete_group_map region remote_group**
Deletes an object group map between regions.

**delete_group_id_map region remote_name**
Deletes user-group name mapping.
delete_region region
   Disconnects a remote region.

delete_role_map region remote_group remote_role
   Deletes a role map within a group map.

delete_user_id_map region remote_name
   Deletes login name mapping.

list_group_id_map region
   Displays mapping from user group names in the remote region to group names in the local region.

list_map region
   Lists object group and role maps between regions.

list_region [region]
   Lists connected regions. If omitted, lists regions that are connected to the local region.

list_user_id_map region
   Displays mapping from user login names in the remote region to login names in the local region.

set_install_pw
   Sets the installation password for creating managed nodes. The old password and the new password are read from standard input. You are prompted for the old and new passwords if you run this command from a terminal device (tty).

   Note: Changing the installation password does not change the region password.

set_region_crypt_level crypt
   Sets encryption level used by other Tivoli servers to connect to this region. The crypt option can be one of none, simple, or DES.

set_region_pw
   Sets encryption password for other Tivoli servers to connect to this region. The old password and the new password are read from standard input. You are prompted for the old and new passwords if you run this command from a terminal device (tty).

   Note: Changing the region password does not change the installation password.

set_allow_rconnect TRUE | FALSE | version_2 [object_dispatcher... | clients | all]
   Enables a Tivoli desktop to connect to a remote managed node. TRUE and version_2 allow remote connections. TRUE uses a private key while version_2 uses a public key. The default is version_2, which is more secure.

set_allow_NAT TRUE | FALSE
   Enables network address translation (NAT) support. You must restart the endpoint manager and gateways for changes to take effect. The default value is FALSE. For more information about NAT support, see the Tivoli Management Framework Planning for Deployment Guide.

set_bdt_port port_value [object_dispatcher... | clients | all]
   Sets the port that the Bulk Data Transfer (BDT) service uses. The port value must be greater than 1023. You can set the port that the service uses for
object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). You must have the super or senior role to run this option.

Note: You can use this option only if the single_port_bdt option is set to TRUE. If set to FALSE (the default), this BDT port is not used and does not apply to any other node operations.

set_comm_check TRUE | FALSE [object_dispatcher... | clients | all]
Sets a communication channel check to determine whether a network connection is still active. The default value is FALSE. When this option is set to TRUE, a Tivoli process that sends data also sends a ping message to the receiving process to verify that the IPC (inter-process communication) channel between the two processes is active. If the IPC channel to the receiving process has been terminated, the resources used by the sending process are freed. Use this option in environments in which systems communicate through firewalls. For security reasons, firewalls can terminate network connections between systems without notifying either system. You must have the super or senior role to run this option.

set_comm_check_response_timeout timeout_value [object_dispatcher... | clients | all]
Sets the number of seconds after which a Tivoli process determines that an IPC channel is terminated. When the odadmin set_comm_check option is set to TRUE, a Tivoli process that sends data also sends a ping message to the receiving process to verify that the IPC channel between the two processes is active. The sending process continues to send data until the set_comm_check_response_timeout value is reached. If the receiving process does not respond to the ping within this timeout, the sending process determines that the IPC channel has been terminated. The default value for this option is 180 seconds and the minimum value is 60 seconds. You must have the super or senior role to run this option.

set_comm_timeout timeout_value [object_dispatcher... | clients | all]
Sets the number of seconds after which a process sends a ping message to determine whether an idle IPC channel has been terminated. When the odadmin set_comm_check option is set to TRUE, a Tivoli process that sends data also sends a ping message to the receiving process to verify that the IPC channel between the two processes is active. This option sets the interval for the ping sent by the process. The process sends the ping only if the IPC channel has been inactive during this interval. The default value for this option is 180 seconds and the minimum value is 60 seconds. You must have the super or senior role to run this option.

set_conn_validation_timeout timeout_value [object_dispatcher... | clients | all]
Sets the number of seconds that the object dispatcher attempts to validate an incoming connection. If the client does not send the correct connection information within this interval, the connection is rejected. The default value is 300 seconds. If you set this value to zero, incoming connection attempts do not time out. You must have the super or senior role to run this option.

set_crypt_level crypt
Sets encryption level for communication between object dispatchers within the local region. Typically, you need to use the following procedure to set the encryption level:
1. Enter odadmin shutdown clients
2. Enter odadmin set_crypt_level crypt
where crypt specifies the encryption level and can be none, simple, or DES.

3. Enter odadmin start clients

You must have the super or senior role to use this option.

set_force_bind TRUE | FALSE [object_dispatcher... | clients | all]
Forces Tivoli communication connections to bind to a single IP address. This option is used in high availability or failover configurations where multiple object dispatchers reside at different IP addresses on a single physical system.

set_install_pw
Sets the installation password for future client installations. The old password and the new password are read from standard input. You are prompted for the old and new passwords if you run this command from a terminal device (tty). You must have the super or senior role to run this option.

set_iom_by_name TRUE | FALSE [object_dispatcher... | clients | all]
Enables or disables communications to rely on the host name rather than the IP address of a Tivoli server when interpreting an IOM key and making a connection. Use this option for multihomed servers that are known by different IP addresses on different subnets.

set_keep_alive [on | off | poll | nopoll | time...]
Sets options for checking for hosts that are down. The default is off, that is, check every 180 seconds to see if a host is still down. See the value referenced in the command output in the line that reads "State checking every 180 seconds". You must have the super or senior role to run this option. The suboptions are as follows:

on | off
Specifies whether to trust cached downed-host information (on) or to try the host each time (off).

poll | nopoll
Specifies whether to poll dispatchers (poll) or collect information implicitly (nopoll). The polling algorithm minimizes network traffic.

time
Specifies the minimum poll interval time in seconds. Most dispatchers are not polled every interval.

The current keep_alive options can be read by running odadmin odinfo 1.

set_network_security [none | SSL | FORCE_SSL] [object_dispatcher... | clients | all]
Sets the network security level of a managed node. You can set the network security level on specified object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). Options are as follows:

none
Specifies that Secure Sockets Layer (SSL) is not used by the managed node, except when it is SSL-capable and is communicating with a FORCE_SSL managed node. This is the default setting.

SSL
Specifies that the managed node uses SSL when communicating with other SSL managed nodes. SSL is not used when communicating to a node with a setting of none.
FORCE_SSL

Specifies that the managed node only communicates using SSL. Non-SSL connections are not accepted by the managed node.

Note: Restart the managed node for changes to take effect. For more information about SSL, see Tivoli Management Framework Planning for Deployment Guide.

set_ORB_pw object_dispatcher

Sets the password for communication between a Tivoli server object dispatcher and the specified object dispatcher on a managed node. The password is read from standard input. You are prompted for the password if you run this command from a terminal device (tty). You must have the super or senior role; you must also have root privileges on the Tivoli server and the specified object dispatcher.

Typically, when a client is added to the Tivoli server, the password used by the Tivoli client to communicate with the Tivoli server is set by the Tivoli server. This password is never stored in a file or transmitted over the network in plain text. However, there is a slight possibility that an intruder could get a copy of this password in its obscured form and decode it if your network is compromised during the installation process or if your database is compromised at a later date.

To change a Tivoli password for a client, do the following:

1. Shut down the affected client.
2. Run odadmin set_ORB_pw object_dispatcher on the Tivoli server.
3. Copy the file host_name-object_dispatcher-odb.adj from the Tivoli database directory to a file named odb.adj in the client database directory. Use a secure copying mechanism to copy this file. Set the file permissions on odb.adj to user-read/write, group-none, and other-none, and set the file ownership to root.
4. Restart the object dispatcher.

set_platform_license license_key

Adds or changes a platform license key. You must have the super or senior role to run this option.

set_port_range [range]

Restricts the IOM channel communication ports and Tivoli Management Framework communication between managed nodes to the specified port range. This option is especially helpful for firewall administrators who need to regulate the availability of ports. The object dispatcher and gateway default ports are not affected by this option. Port ranges must be greater than 1023. To set the port range to null, use the following syntax:

odadmin set_port_range ""

set_rpc_max_threads num_threads

Sets the concurrent remote procedure call thread limit for the dispatcher or the number of concurrent object call threads. The default is 250.

set_ssl_ciphers cipher... [object.Dispatcher... | clients | all ]

Sets ciphers for managed nodes to protect the channel. You can set ciphers on specified object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). Values for the cipher options are as follows:
default
  Specifies the default Tivoli list of 05040A030609. A node can be set to the default regardless of its SSL capabilities.

cipher
  Specifies the cipher list, in order of preference (for example, 0A09). The node must be SSL-capable before you can change a user-defined cipher list. For cipher values and information about how to enable SSL on a managed node, see Tivoli Management Framework Planning for Deployment Guide.

Note: Restart the managed node for changes to take effect.

set_tmgr_retries
  Adjusts the default timeout value for the transaction manager. The transaction manager retries messages once per minute, so that the number of retries is equal to the number of minutes that it should wait before aborting a transaction.

Note: This value is adjusted on a per Tivoli region basis; every dispatcher in the region gets the same value. The new value takes effect the next time a dispatcher is restarted.

shutdown [object_dispatcher... | clients | all]
  Shuts down the local object dispatcher. You can shut down specified object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). You must have the super or senior role to run this option. You cannot shut down the Tivoli server from a remote system. This option is not available to NetWare clients. NetWare clients must be stopped with the oservend command.

single_port_bdt [TRUE | FALSE] [object_dispatcher... | clients | all]
  Enables or disables single-port Bulk Data Transfer (BDT)—the underlying support for Inter-Object Messaging (IOM) and other large-scale data transfers. If TRUE is specified, the port usage is consolidated to the selected port. The default is FALSE. You can enable or disable the service for object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). You must have the super or senior role to run this option.

Notes:
  - After you enable BDT service, use the reexec option to recycle the object dispatchers.
  - If you enable single_port_bdt, you can change from the default port (9401) using the set_bdt_port option.

start [object_dispatcher... | clients | all]
  Starts the local object dispatcher. You can start specified object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). You must have the super or senior role to run this option. This option is not available to NetWare clients. NetWare clients must be started with the oservrun command.

trace objcalls | services | errors | off [object_dispatcher... | clients | all]
  Starts or stops debug tracing. You can specify that object calls, services, or errors be traced, or you can turn off debug tracing. If you want to trace object calls, services, and errors, you must turn first on the tracing of errors by using the odadmin with the trace errors option. Note that turning error
tracing on also turns off the tracing of object calls and services. Therefore, ensure that you turn object call and services traces on after the error trace invocation.

You can start or stop tracing for specified object dispatchers (object_dispatcher), all object dispatchers on managed nodes (clients), or all object dispatchers (all). If you do not specify specific clients or the clients or all option, debug tracing is started or stopped on the local object dispatcher. The trace information is collected in the odtrace.log file in the database directory. You can use the wtrace command to view this trace information. You must have the super or senior role to run odadmin command with the trace option.

Tivoli recommends that you do not enable error tracing by default. Tracing all object calls and services impacts the performance of your dispatcher. Use tracing of all object calls and services only when diagnosing specific problems.

Examples

1. The following example gets the status of the Tivoli server:
   
   odadmin odinfo 1
   
   Tivoli Management Framework (tmpbuild) #1 Wed Sep 17 14:07:45 CDT 2003
   (c) Copyright IBM Corp. 1990, 2003. All Rights Reserved.
   
   Region = 1849216852
   Dispatcher = 1
   Interpreter type = aix4-r1
   Database directory = /data/lorozco/Tivoli-1234/db/liliana.db
   Install directory = /data/lorozco/Tivoli-1234/bin
   InterDispatcher encryption level = simple
   Kerberos in use = FALSE
   Remote client login allowed = TRUE
   Install library path = /data/lorozco/Tivoli-1234/lib/aix4-r1:
   /usr/lib:/data/lorozco/tmp_install/iblib/aix4-r1:/usr/lib
   Force socket bind to a single address = FALSE
   Perform local hostname lookup for IOM connections = FALSE
   Use Single Port BDT = FALSE
   Use communication channel check = FALSE
   Communication check timeout = default (180 secs)
   Communication check response timeout = default (180 secs)
   Oserv connection validation timeout = 300
   Port range = (not restricted)
   Single Port BDT service port number = default (9401)
   Network Security = none
   SSL Ciphers = default
   ALLOW_NAT = FALSE
   State flags in use = TRUE
   State checking in use = TRUE
   State checking every 180 seconds
   Dynamic IP addressing allowed = FALSE
   Transaction manager will retry messages 4 times.
   
2. The following example gets help on the shutdown option:
   
   odadmin help shutdown
   
3. The following example flushes the database files for all dispatchers:
   
   odadmin db_sync all
   
4. The following example specifies that all managed nodes use port range 60000 through 60100:
   
   odadmin set_port_range 60000-60100
5. The following example lists information about the all object dispatchers in a Tivoli region:

```
  odadmin odlist
```

<table>
<thead>
<tr>
<th>Region</th>
<th>Disp</th>
<th>Flags</th>
<th>Port</th>
<th>IPaddr</th>
<th>Hostname(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1248901349</td>
<td>1</td>
<td>ct-</td>
<td>94</td>
<td>10.69.9.42</td>
<td>la.tivoli.com,la</td>
</tr>
</tbody>
</table>
| 2     | ct-  | 94    | 10.69.9.73 | ten.tivoli.com |}

---

**See Also**

odbls, odstat, oserv, wconnect, wdisconn, wlsconn, wrmnode, wtrace, wupdate
odbls

Lists the contents of an object database.

Syntax

```
```

Description

The `odbls` command lists the contents of an object database.

Options

- `–a` Displays the attributes in the object database.
- `–i` Displays the inheritance trees in the object database. To use this option, you must use the Tivoli server database.
- `–l` Displays the object database inheritance list.
- `–k directory` Specifies the directory that contains the object database to be listed. If omitted, the database in the current directory is listed.
- `–l` Displays a verbose listing of the requested information.
- `–m` Displays the method headers and dumps all entries. To use this option, you must use the Tivoli server database.
- `–M method_name` Searches through method headers and dumps entries for `method_name`. To use this option, you must use the Tivoli server database.
- `–O` Walks through the object database. This is the default.
- `–s` Forces the appropriate object dispatcher to update the database that is to be listed. This synchronization ensures that the command reports the same data that the object dispatcher is using. If omitted, no synchronization is performed before listing the object database contents.
- `oid` Restricts the listing to the specified object.

Authorization

You must have read permission on the database to use the `odbls` command.

To use the `–s` option, `super`.

Examples

The following example lists all objects in the object database:

```
odbls -k /var/spool/tivoli/myhost.db
```

<bootstrap>
200003.0.0
200003.1.0
200003.1.1
200003.1.10
200003.1.100
200003.1.101
200003.1.102
See Also

odadmin
odstat

Lists the status of current and recent object calls.

Syntax


Description

The odstat command lists the status of current and recent object calls for the specified object dispatcher. This command can list object calls from a running dispatcher.

The first form of the command collects the object dispatcher history by invoking a method. You can specify a remote object dispatcher if desired. The second form of the command collects information by sending a signal to the object dispatcher.

The -k option specifies the database directory; the object dispatcher dumps its information to a temporary file in this directory. The pid option specifies the object dispatcher to send a signal to. The second form of the command is useful when communication is disrupted for some reason, but is more specific than the first form. The second form can only send signals to local processes, and because the object dispatcher is owned by Administrator or root, you can only run this form if you are logged in as Administrator or root.

The command returns its output in the following columns:

| tid  | Lists the thread ID. When you start an object call, two threads are generated: one for the object call itself and one for the method being invoked. |
| type | Lists the thread type. The thread type flags are as follows: |
|      | O Specifies the object call thread (attached to an object request). This indicates that the method was invoked here but is running elsewhere. |
|      | M Lists the method thread (attached to a method). The object call occurred on a different system, but the object is located on the local system. |
|      | O+ Indicates that the object call and method threads are the same. This indicates that the caller and method are both local. |

The method type flags are as follows:

| a | Asynchronous object call |
| b | One-way object call |
| d | Daemon method |
| h | Helperless method |
| o | Per-object method |
| q | Queueing method |

| ptid  | Lists the parent thread ID or the thread ID of the object call whose method |
made the current object call. If this field is blank, the object call is external. The number before the dash (-) is the dispatcher number where the parent thread resides. The number after the dash is the thread ID in the parent object dispatcher.

**State** Lists the following states of the object call thread:

- **ali**: The thread is performing a lookup on the Tivoli object database.
- **coord**: The method is serving as a transaction coordinator.
- **done**: The object call is complete.
- **err**: An internal error terminated the thread.
- **init**: The thread has been initialized.
- **mwait**: The thread is waiting for the associated method thread to complete.
- **rwait**: The thread is waiting for the caller to collect the results of an asynchronous object call.

The states of method threads are as follows:

- **done**: The method is complete.
- **gmeth**: The thread is obtaining the method code from another dispatcher.
- **hdwt**: The thread is waiting for the daemon-method process (nonqueueing daemon) to be ready to accept another request.
- **init**: The thread has been initialized.
- **run**: The method is running.
- **serv**: The thread is performing an object services call.
- **twait**: The method is waiting on transaction status to commit or abort.

**StdO** Lists the number of bytes written to standard output by the method.

**StdE** Lists the number of bytes written to standard error by the method. Most threads do not write to standard error.

**Start** Lists the time that the thread started. Entries can be one of the following depending on the age of the thread:

- Time–The thread started on this day.
- Day and Time–The thread started before this day, but within the current week.
- Month and day (UNIX operating systems only)–The thread started before the current week.
- Month (Windows operating systems only)–The thread started before the current week.

**Err** Lists the error status of the thread. If this field is blank, no error occurred. Error types are as follows:

- **e=n**: The method returned n as its exit code. Error codes 0 through 21 are reserved for system-defined errors. Application error codes start at 22.
- **s=n**: The method failed due to signal n.
- **s=n**: The method failed due to signal n and produced a core file. Contact your Support provider if you want to debug using this core file.
**odstat**

XXX An uppercase word indicates an error in the object dispatcher.

Exit codes (=) can come from the system on which the Tivoli desktop is running, Tivoli Management Framework, or an application. It might be necessary to look at different sets of error documentation to find out which is the most likely source. You might be able to use system documentation to obtain help regarding system-produced errors. Most UNIX operating systems include an error file that lists the system error codes and often a short description. On OS/2 operating systems, enter `help n`, where `n` is the number of the error message (for example, `help 5`). On Windows operating systems, enter the `net helpmsg n` command (for example, `net helpmsg 5`).

**Method**

Lists the text of the method invocation. The first value is the object ID of the object in whose context the method was invoked. This can take the form of three numbers separated by periods or three numbers separated by periods followed immediately by a pound sign (#), the class name, and another pound sign. The next entry is the name of the method followed by the method options.

The following is an example of a method listed in the odstat output:

```
1242184237.1.516#TMF_SysAdmin::InstanceManager# _get_interfaces
```

For more information about command output and other problem determination commands, see *Tivoli Management Framework Maintenance and Troubleshooting Guide*.

**Options**

? Displays help for the command.

-a Lists all threads. By default, system threads are omitted.

-c Lists currently running threads.

-d Lists the active method-daemon processes.

-h Lists thread history (terminated threads) only.

-k `dbdir` (UNIX operating systems only)

Returns information from the dispatcher that is using the database in the specified directory. You should specify the object dispatcher process ID (PID) if you specify the `-k` option. If you do not, the command randomly picks the object for the dispatcher process ID.

-l Returns a long listing.

-o `baseoid`

Returns object dispatcher status from another system.

-p `port_no`

Sets the port number.

-s Returns a short listing.

-v Specifies verbose mode. Lists the last object-services request that the method or thread made, that request return code, the process ID associated with the method, and the last method invoked in a daemon thread.

pid Specifies the process ID of the dispatcher. If you specify this option, you must also specify the `-k` option.

If no options are specified, the command default is *odstat* `-c -h -l -o 0.0.0`. 
Authorization

You must have the super, senior, admin, or user role to run this command. You must be Administrator or root to run the -k option.

Examples

1. The following example shows the command output when no options are specified. Explanations are included throughout the example.

   odstat

   The output shows the methods currently running:
   
   n_active = 5 n_free = 195
   tid type   ptid State StdO StdE  Start  Err       Method
   83 O+bhdq  run  0  0 Sat16:00 200003.1.163#TMF_Scheduler::scheduler# start

   In the previous section of the output method thread, ID 83 is a “one way” invocation that invoked a helperless, queueing, daemon, per-object method implementation. It is currently in a run state, and started at 16:00 on Saturday. The object ID is 200003.1.163#TMF_Scheduler::scheduler# (the local scheduler object), and the method name is start.

   (... output deleted for brevity ...)

   ---- history ----
   855 O+ 1-854 done 11 0 Sun16:16 0.0.0 get_name_registry
   856 O+bhdq 1-854 done 106 0 Sun16:16 200003.1.26 lookup
   857 O+bhdoq 1-854 done 6 0 Sun16:16 200003.1.128
   #TMF/UI::ActiveDesktopList# add_entry

   The history line indicates the beginning of methods that have already completed. Methods 855, 856, and 857 are cascaded methods; they were invoked by method 854 on this oserv. (This example uses dispatcher 1.) Method 855 produced 11 bytes of standard output, 856 produced 106 bytes, and 857 produced 6 bytes.

   (... more output deleted ...)
   * 918 O+bhdq 1-917 done 488 0 Sun16:27 e=12 200003.1.26 lookup

   Method 918 produced an error. The asterisk in the first column indicates a possible abnormal condition. The e=12 indicates an exception occurred. The wtrace command can be used to determine more information about the exception.

   950 0 1-949 done 0 0 Sun16:28 <batch-mgr>
   951 O+bhdq 1-949 done 117 0 Sun16:28 200003.1.378
   #TMF_Install::ProductInfo# add_backref_optimized

   Method thread 950 is a batch-object-call manager. It indicates that there were multiple invocations of the same method on multiple objects at the same time. Line 951 is one of the batched method invocations managed by 950.

   * 1029 O+ 1-1026 done 0 0 Sun17:06 UNAUTHORIZED 200003.0.0
      get_principal_roles Root_PI-slugo

   Method 1029 shows a method that ran and failed at the core services level, below the exception facility. In this case, odstat translates the error code to an error name, where possible. Because an exception did not occur, wtrace is not likely to produce additional information about the error code.

2. The following example queries the method history on a remote machine. First, use the name registry wlookup command to translate the machine name to an object reference, and then call odstat with that object reference.
odstat

wlookup -r ManagedNode -a

pokey 200003.2.7#TMF_ManagedNode::Managed_Node#

sluggo 200003.1.285#TMF_ManagedNode::Managed_Node#

odstat -o 200003.2.7#TMF_ManagedNode::Managed_Node#

See Also

odadmin | tmstat | wtrace
oinstall

Installs, updates, or removes the Tivoli object dispatcher service in a Windows Service Manager.

Syntax

oinstall –install path

oinstall –query

oinstall –remove

oinstall –update [+auto | –auto] [+depend | –depend service] [+interactive | –interactive] [path]

Description

The oinstall command is used to install the Tivoli object dispatcher in the Windows Service Manager. This command also is used to update the object dispatcher or to remove it from the Windows Service Manager. When you install Tivoli Management Framework on a Windows operating system, this command runs automatically during the installation process.

Options

–install
Installs the object dispatcher service in the Windows Service Manager. The service is installed in noninteractive mode and manual restart mode.

–query
Enables you to see the dependencies associated with the current object dispatcher service. This option also returns status of the object dispatcher installation.

–remove
Removes an object dispatcher service from the Windows Service Manager.

–update
Updates the object dispatcher with one or more of the following options:

+auto | –auto
Enables or disables automatic restart when you restart your system. By default, automatic restart is disabled.

+depend | –depend service
Delays the start of the object dispatcher on a Windows operating system until the specified service is started. Services include TCPIP, TRIP, and so on. If the specified service is down and you attempt to start the object dispatcher, the system tries to start the service before starting the object dispatcher. If you try to stop the specified service, you are prompted with a dialog stating that the object dispatcher is dependent on this service. If you proceed to stop the service, the object dispatcher also is stopped. You can also remove an object dispatcher service dependency by using this option. If the service is running when you issue the update, the service is stopped and must be restarted for the change to take affect. The dependent service then returns a prompt about stopping the object dispatcher service.
oinstall

+interactive | –interactive
Enables or disables the interactive mode between the object dispatcher service and the Tivoli desktop. Unless you are doing Tivoli ADE (Application Development Environment) development, do not enable the interactive mode.

path
Specifies the path to the oserv.exe executable on a local New Technology File System (NTFS). It is assumed that the libuthreads.dll file is located in the same library.

Note: If the service is running when you issue an update command, you must stop the service and restart it for changes to take effect.

Authorization
Log on as service privilege

Examples

1. The following example installs the object dispatcher service in the Service Manager. The oserv.exe and libuthreads.dll files are located in c:\Tivoli.
   oinstall -install c:\Tivoli\oserv.exe

2. The following example updates the object dispatcher service in the Service Manager. The location of the oserv.exe and libuthreads.dll files is changed to c:\Tivoli\bin and the automatic restart mode is enabled.
   oinstall -update +auto c:\Tivoli\bin\oserv.exe

3. The following example delays the start of the object dispatcher on a Windows operating system until TCP/IP is started:
   oinstall -update +depend TCP/IP

4. The following example removes the object dispatcher service dependency:
   oinstall -update -depend

5. The following example enables you to see what dependencies are associated with the current service:
   oinstall -query
oserv

Provides operations to control and configure object dispatchers.

Syntax

```bash

oserv -k dbdir [-a TRUE | FALSE] [-B libpath] [-d] [-E [default | cipher...]] [-h

oserv -k dbdir [-f TRUE | FALSE] [-i] [-r region] [-a TRUE | FALSE]
swap_size] [-n [none | SSL | FORCE_SSL]] [-N [by_addr | by_name | ali]] [-p

oserv -k dbdir [-f TRUE | FALSE] [-i] [-h Tivoli_server] -b install_dir [-B libpath] [-d]
[-E [default | cipher...]] [-I log_file] [-m swap_size] [-n [none | SSL | FORCE_SSL]]
[-p local_port] [-S] [-s install_key] [-t max_trace_size] [-v] [-z TRUE | FALSE]
```

Windows systems only

```bash
net start oserv /-option...
```

The `oserv` command can only be invoked from the Service Manager. From the
command line, the `oserv` command must be preceded by `net start`.

All options must be preceded by a slash and a dash (`/`) instead of the single dash (`-`) used in UNIX. Any letter options that require a value cannot have a space
between the letter and the value (for example, `net start oserv /-Nbyaddr`).

The `-k` option is not required. If not specified, the location of the database
directory is taken from the registry.

Description

The `oserv` command starts the Tivoli object dispatcher. The Tivoli object dispatcher
has multiple functions. It maintains the Tivoli object database on each system that
has Tivoli installed, it routes object calls to the proper systems and objects, and it
arranges for the execution of methods that are invoked in the context of objects
that reside on the local system.

The first form of the `oserv` command is used to restart the object dispatcher on a
Tivoli server.

The second form of the `oserv` command is used to restart the object dispatcher on
a managed node.

The third form of the `oserv` command is used to initialize the object dispatcher on
a Tivoli server.

The fourth form of the `oserv` command is used to initialize the object dispatcher on
a managed node.
The third and fourth forms of the `oserv` command are used only when you are using the installation programs.

The object dispatcher saves its configuration options in its database so that the object dispatcher resumes with its previous settings when it is restarted. When restarting an object dispatcher, you typically specify the `-k` option plus the options for any settings you want to change. You can change the object dispatcher settings without shutting down the object dispatcher by using the `odadmin` command.

**Options**

`-a TRUE | FALSE`  
Sets the flag that controls client installations. Client installations can be allowed (TRUE) or prohibited (FALSE). This option is only valid for the Tivoli server.

`-b install_dir`  
Specifies the path name of the method binaries installation directory. This option is required if the `-i` option is specified. Note that `$BINDIR/..` and `%BINDIR%\..` are literal path name values, and the two dots after the slash are part of the name. The dots are not an ellipses (...) indicating a continuation of a path name. This is the same directory referred to by `$BINDIR/..` (UNIX operating systems) and `%BINDIR%\..` (Windows operating systems).

`-B libpath`  
Specifies the shared library search path. If the `-i` option is specified and this option is not specified, the library search path is read from the local environment. This is the same directory referred to by `$LIBDIR` (UNIX operating systems) or `%LIBDIR%` (Windows operating systems).

`-c crypt`  
Specifies the intraregion encryption level. The `crypt` option can be `des`, `simple`, or `none`.

`-C crypt`  
Specifies the interregion encryption level. The `crypt` option can be `des`, `simple`, or `none`.

`-d`  
Does not detach the object dispatcher from the controlling terminal. This option is used for running an object dispatcher in a debugger. Standard output is not mapped to the `/dev/null` directory.

`-E [default | cipher...]`  
Sets one or more ciphers for managed nodes, in order of preference (for example, 0A09). This option sets ciphers for the initial connection of a managed node to the Tivoli server. After the node connects, it defaults back to the value stored on the server. You also can specify that a managed node has the default setting, which means that the managed node has the default Tivoli cipher list of 05040A030609. For a list of cipher values, see *Tivoli Management Framework User’s Guide*.

`-f TRUE | FALSE`  
Forces an object dispatcher to bind to a single socket for all managed node connections. Use this option when multiple dispatchers are running on a machine that has multiple Internet Protocol (IP) addresses. This forces each dispatcher to use the same IP address each time it communicates with clients.
-h Tivoli_server
   Specifies the name of the new Tivoli server. This option is used when
   starting the object dispatcher on a managed node.

-i
   Initializes the object dispatcher. This option is used the first time the object
   dispatcher is started on a system. Do not use the -i option without
   assistance from Customer Support.

-I
   Indicates to the object dispatcher that the object dispatcher is being started
   by the inetd service. This option is only specified in inetd configuration
   files; it should not be specified on the command line.

-k dbdir
   Specifies the path name of the object database directory. This is the same
   directory referred to by $DBDIR (UNIX operating systems) or %DBDIR% (Windows operating systems).

-l log_file
   Specifies the file to be used for logging messages. If this option is not
   specified, messages are logged to the oservlog file.

-m swap_size
   Specifies the size of the swap space allocated by the mmap command.

-n [none | SSL | FORCE_SSL]
   Sets the network security level of a managed node’s initial connection to
   the Tivoli server. After the node connects, it defaults back to the value
   stored on the server. Options are as follows:

      none  Specifies that SSL is not used by the managed node, except when it
            is SSL-capable and is communicating with a FORCE_SSL managed
            node. This is the default setting.

      SSL   Specifies that the managed node uses SSL when communicating
            with other SSL managed nodes. SSL is not used when
            communicating to a node with a setting of none.

      FORCE_SSL
            Specifies that the managed node only communicates using SSL.
            Non-SSL connections are not accepted by the managed node.

-N [by_addr | by_name | ali]
   Refetches IP addresses for the odlist entries using the gethostbyaddr
   (by_addr), gethostbyname (by_name), or by replacing the odlist entry of
   the Tivoli server with the name and address of this system (ali). Do not
   move your Tivoli server to a new system by using the ali option without
   first getting instructions for this procedure from Customer Support.

-p local_port
   Specifies the port number to use for communication with other processes.
   The port number must be less than 1024. This option overrides the port
   specification in the /etc/services file. If this option is not specified, the
   default is the port specification in the /etc/services file.

-P ali_port
   Specifies the port number to use for communication with the Tivoli server.
   The port number must be less than 1024. This option is required only if the
   -i option is specified and the Tivoli server port is different than the port
   for the local object dispatcher. This option is valid only for managed nodes.

Note: This option is meant for use in development and test environments
only. It should not be used in a production environment.
–r region
   Sets the Tivoli region number. This option can be used only with the –i option or the –N ali option.

–R irkey
   Specifies the interregion encryption key. If the irkey option is not specified, the interregion encryption key is read from standard input. You cannot specify the –R option when the –s option is specified.

–s install_key
   Specifies the installation key. If the install_key option is not specified, the installation key is read from standard input. You cannot specify the –s option when the –R option is specified.

–S
   Suppresses the output of errors to the syslogd service.

–t max_trace_size
   Sets the maximum size of the odtrace.log file created for use by the wtrace command.

–v
   Causes the object dispatcher to use the vfork() method to spawn subprocesses instead of fork() methods on UNIX operating systems. Specifying this flag has no effect on Windows operating systems.

Authorization
   root

Examples
   1. The following example starts the oserv, using all current defaults:
      
      oserv -k /var/spool/Tivoli/myhost.db

   2. The following example changes the path to the binary and library directories. This is helpful if the mount point to these directories is changed.
      
      oserv -b /mnt/local/Tivoli/bin -B /mnt/local/Tivoli/lib:/usr/lib
      -k /var/spool/Tivoli/myhost.db

See Also
   odadmin, odstat, idlcalf, objcall, oinstall, wsettap, wtrace
tivoli

Starts the Tivoli desktop and previews a new dialog.

Syntax

tivoli [-debug] [-display display] [-help] [-host host_name] [-port port_number] [-preview file.d variable_name variable_value...] [-user user_name] [x_options]

Description

The tivoli command performs one of two different kinds of functions, depending on the options used. First, it can start the Tivoli desktop for a Tivoli administrator. Second, it can preview a dialog created with DSL programming language.

The -display option specifies the X window display on which to display the desktop or preview the dialog. If this option is omitted, the tivoli command defaults to the value defined in the DISPLAY environment variable. See the Tivoli Enterprise Installation Guide for more information about X window resources.

The values of the x_options option are strings to be interpreted as X resource settings, for example, -background blue sets the default background color of all dialog backgrounds to blue.

If the -preview switch is omitted, the tivoli command starts the administrator desktop.

The -display and x_options options are not used for Windows systems.

For Tivoli Application Development Environment (ADE) developers, if the -preview option is specified, it instructs the User Interface (UI) server to preview the dialog contained in file.d. If the dialog contains dialog variables, the name and value of each dialog variable must follow the dialog file name. For example, if the dialog has two dialog variables named var1 and var2, the command must be invoked in the following way:

tivoli -preview file.d var1 value-of-var-1 var2 value-of-var-2

If the value contains blanks, it needs to be enclosed in double quotation marks and single quotation marks, for example:

tivoli -preview file.d var1 "'value of var 1'"

Options

- debug
  Displays an additional dialog that contains debugging information for ADE developers.

- display display
  Specifies to display the desktop on the screen of the host specified in display.

- help
  Prints a usage message.

- host host_name
  Specifies the managed node, including the Tivoli server where the Tivoli desktop should connect.
-port port_number
  Specifies the port number used by the object dispatcher.

-preview file.d variable_name variable_value
  Instructs the UI server to preview the file specified by file.d. This option can be used by ADE developers to preview new dialogs before they are built into a product. Suboptions are as follows:

  file.d  Specifies the name of a .d file for ADE developers.

  variable_name
    Specifies a Dialog Specification Language (DSL) variable name.

  variable_value
    Specifies the value associated with variable_name.

-user user_name
  Specifies the login name to the managed node.

x_options
  Specifies the X resources to set for this session.

Authorization
  user, admin, senior, super
tmcmd

Forces a change of state of a running transaction.

Syntax

```
  tmcmd [-p port] state transaction_id...
```

Description

The `tmcmd` command forces a transition of a transaction state. This command sends a message to the local transaction manager to force transactions into the specified state.

The `tmcmd` command is primarily useful when testing and debugging the transaction manager, which is linked inside the oserv service, or daemon. Customer Support may ask you to run this command to recover from a severe or unusual problem as a last resort. It is strongly recommended that this command only be run with the direction and assistance of Customer Support, as this command can crash your oserv daemon or corrupt your database if used inappropriately.

Options

```
  -p port
    Specifies the local port number.

  state
    Specifies the state to force the transactions to. This option can be one of abort, commit, prepare, complete, prepared, or completed.

  transaction_id
    Specifies the ID of the transaction for which a state change is to be forced. You can specify more than one transaction ID.
```

Authorization

```
  senior, super
```

Diagnostics

If the command sent to the transaction manager violates the two-phase commit protocol, the oserv daemon may abort or your database could be corrupted.

See Also

```
  odstat tmstat
```
tmstat

Displays the status of current transactions and locks.

Syntax

```
tmstat [-k dbdir] [-p port] [-r region] [-v] [-a] [base_object_id...]
```

Description

The tmstat command displays the currently running transactions and locks and their current state. This command is primarily a debugging tool for users who are developing transaction-based applications; it allows such users to observe their transaction hierarchy.

Each transaction ID that is displayed implicitly contains the transaction hierarchy; you can interpret {transA}{transB} to be a child of {transA}.

Options

- `-a` Displays all object IDs in the base list of the local Tivoli region or the remote Tivoli region specified by the `-r` option.
- `-k dbdir` Specifies the database directory.
- `-p port` Specifies the local port number.
- `-r region` Queries a different Tivoli region. This option specifies the base object ID on the remote Tivoli server.
- `-v` Specifies verbose mode. Lists of subtransactions are dumped.

`base_object_id...`

Specifies the object dispatcher to query. Multiple `base_object_id` options can be specified.

Authorization

user, admin, senior, super

Examples

1. The following example shows the output for no options. An explanation of the output follows the example.

```
    tmstat
    Transactions for 0.0.0
    Trid Type State Resources Polling Coord Parent MId
    ----------------------------------------------------------
    {202020:1,202020:1,2:3} Top-T running Yes No running running 40
    {202020:1,202020:1,2:3},{202020:1,202020:1,2:4} Sub-T commit Yes running running 44
    {202020:1,202020:1,2:3},{202020:1,202020:1,2:5} Sub-T commit Yes running running 45
    {202020:1,202020:1,2:3},{202020:1,202020:1,2:6} Sub-T commit Yes running running 46
    {202020:1,202020:1,2:3},{202020:1,202020:1,2:7} Sub-T commit Yes running running 47
    ----------------------------------------------------------
```

Tivoli Management Framework: Reference Manual
The output of **tmstat** consists of four sections: a table listing the current transactions, a section listing coordinators, a section stating why the transaction log file cannot be deleted, and a section listing the remote transactions with local resources.

The previous example shows only two of the four possible sections. The table lists the transaction IDs of all local transactions, the transaction type, the transaction state, whether the transaction holds resources, whether the transaction is polling its parent or coordinator, and the state the transaction believes its parent or coordinator to be in. The MTid entry can be used to match a transaction ID with a method thread ID in the **odstat** command.

The second section, listing coordinators, was empty and was skipped.

The third section indicates any current reasons why the transaction log file cannot be deleted. If this section is empty, the **odadmin db_sync** command can be used to truncate the log file immediately.

The fourth section, listing remote transactions with local resources, was empty and was skipped.

2. The following example invokes the **tmstat** command on a remote managed node. For example, from the Tivoli server, you could display the status of transactions and locks on “dispatcher 2” by following these steps:

   a. To determine the OID of the managed node in which you are interested, enter the following command:

   ```bash
   wlookup -ar ManagedNode
   ```

   Output similar to the following is displayed:

   ```bash
   bushido 1248901349.1.348#TMF_ManagedNode::Managed_Node#
tengu 1248901349.2.7   #TMF_ManagedNode::Managed_Node#
   ```

   b. To invoke the **tmstat** command using the explicit OID of the managed node, enter the following command (where **1248901349.2.7** is the object ID):

   ```bash
   tmstat 1248901349.2.7
   ```

   Or you can use the **$TMR** variable to substitute for the unique Tivoli region identifier number as shown:

   ```bash
   tmstat $TMR.2.7
   ```

---

**See Also**

- **odstat**
- **tmcmd**
vdisp

Checks status of all managed nodes.

Syntax

vdisp [-v] [-h] [-i] [-q]

vdisp [-v] [-h] [-i] [-t time]

Description

The vdisp command checks the status of all managed nodes listed in the object dispatcher list and optionally performs a simple performance test.

Note: Use this command only as directed by your Support provided.

Options

- **-h** Displays host and status information about each managed node in the Tivoli region.
- **-i** Ignores managed nodes that are not communicating with the object dispatcher. If this option is not specified and there are managed nodes that are down, the command stop processing.
- **-q** Does not perform the simple performance check. This option overrides the -t option.
- **-t time** The amount of time in seconds to perform the performance test. The default is 30 seconds. This option is ignored if the -q option is specified.
- **-v** Performs a ping and performance test against all managed nodes in the object dispatcher list.

Examples

1. The following example pings all managed nodes in the object dispatcher list, displays the host status of each, ignores any that are offline, and does not perform a performance check.

   vdisp -vhiq

   ping:
   1234567890.1.0 alive
   1234567890.2.0 alive
   ORB  ServConn     Host     Type
   1   UP   paper.tivoli.com  solaris2
   2   UP   vellum.tivoli.com  solaris2

2. The following example pings all managed nodes in the object dispatcher list, ignores any that are offline, and performs a 5 second performance test:

   vdisp -vit5

   ping:
   1234567890.1.0 alive
   1234567890.2.0 alive
   Running performance test for 5 seconds
   ORB  Ncalls Secs/Call Calls/Min
   1234567890.1.0  2020  0.05  1218.06
   1234567890.2.0  2013  0.05  1214.07
w4inslcf.pl

Installs an AS/400 endpoint on an iSeries system.

Syntax


Description

The \texttt{w4inslcf.pl} command installs and optionally starts an AS/400 endpoint daemon job on one or more iSeries systems. You can install multiple endpoints by listing the system name on the command line or supplying a file containing the system names. The file must contain one system name per line.

The command checks for prerequisites, sends code to the endpoint to be installed, using FTP, and then restores the product, ITMELCF, to the AS/400, using the \texttt{RSTLICPGM} command. If requested, the endpoint is also started by issuing the AS/400 \texttt{STRTMEEPT} command.

Options

\texttt{\textbf{-a}} \hspace{1em} Specifies asynchronous installation of endpoints. By default, the command waits for the endpoint to log in to its gateway before installing the next endpoint.

\texttt{\textbf{-f file_name}} \hspace{1em} Specifies the file containing a list of systems on which to install endpoints. This file contains one system name per line, specifying the user ID and password to be used. Each line in this file must be in the following format:

\begin{verbatim}
host [user ID] [password]
\end{verbatim}

\texttt{\textbf{-F}} \hspace{1em} Forces an overwrite of an existing installation.

\texttt{\textbf{-g gateway_label[+port]}} \hspace{1em} Specifies the IP address or host name and optionally the port number of the gateway to which the endpoint logs in.

\texttt{\textbf{-I}} \hspace{1em} Indicates that the endpoint should be installed but not started.

\texttt{\textbf{-l endpoint_port}} \hspace{1em} Specifies the port number for the endpoint. The default port number is 9495.

\texttt{\textbf{-L config_options}} \hspace{1em} Passes configuration options to the daemon for starting the endpoint. To pass multiple options, enclose them in quotation marks. Refer to the \texttt{lcfd} command for a list of valid options.

\texttt{\textbf{-N code}} \hspace{1em} Specifies additional languages to support by AS/400 code. The following are the codes: 2980 (Brazilian, Portuguese), 2989 (Chinese, simplified), 2924 (English uppercase and lowercase), 2938 (English uppercase), 2928 (French), 2929 (German), 2932 (Italian), 2962 (Japanese), 2986 (Korean), and 2931 (Spanish). To specify multiple languages, enclose them in quotation marks. The defaults are 2924 and 2938.
--P Specifies to prompt the user for a password. This option overrides existing entries in a $HOME/.netrc file used for automatic logins.

--s dir_name Specifies the directory that contains the endpoint installation image. This directory can be on a compact disc, the Tivoli server, a gateway, or a network drive.

--T ip_address Specifies which IP address on the installing managed node to use for the installation. This option is required only when the machine running the command has multiple network interface cards (NICs) and you want to install using a network interface card other than the default.

--v Writes verbose messages to standard output. Error messages are still written to standard error.

--Y Specifies that the installation should proceed without confirmation. The default is to request confirmation.

eendpoint... Specifies the names or IP addresses of iSeries systems on which the endpoints are installed.

Authorization
You must have root access to install endpoints, but you do not need any Tivoli authorization roles. On the iSeries system, you need authority to use the RSTOBJ and RSTLICPGM commands and *SAVSYS special authority.

Return Values
Returns one of the following:

0 Indicates successful completion.

-1 Indicates failure due to an error.

Note: Host names for failed installations are written to the os4LcfH.err file and can be retried with the command using the -f os4LcfH.err command option.

Examples
1. The following example installs the AS/400 endpoint on the iSeries systems as41.tivoli.com and as42.tivoli.com, connects to gateway smithers.tivoli.com using port 9494. The installation image is located in the /cdrom/1tmelcf directory.

   w4inslcf.pl -v -g smithers.tivoli.com+9494 as41.tivoli.com
   as42.tivoli.com -s /cdrom/1tmelcf

2. The following example installs the AS/400 endpoint on the iSeries system as42.tivoli.com, connects to gateway smithers.tivoli.com using port 9494, and installs French and German language support.

   w4inslcf.pl -v -g smithers.tivoli.com+9494 -N '2928 2929'
   as42.tivoli.com

3. The following example causes the installing computer system to use the network interface card with IP address 146.84.36.37 to install the endpoint on the iSeries system cheese.dev.tivoli.com, instructs the AS/400 endpoint to log in to the gateway using IP address 146.84.36.42 and port number 8752, and causes the gateway to use IP address 146.84.39.126 and port 18752 to communicate with the endpoint. The endpoint label is ques.0
w4inslcf.pl -T 146.84.36.37 -g 146.84.36.42+8752 -l 18752
-L 'local_ip_interface=146.84.39.126 lcs.machine_name=queso'
cheese.dev.tivoli.com
waddicon

Adds an icon to a Windows Program Manager group.

Syntax

```
waddicon -g group_name [-a] [-c "command_line"] [-i icon_file] [-m "message"] [-r]
[-t icon_title]
```

Description

The `waddicon` command adds a new icon to a Windows Program Manager group. If the group does not exist, it is created. If the command is launched as a batch from the Windows service agent, the created program group is a common program group. If the command is launched from the Windows console agent, the created program group is a user program group.

Options

- `–a` Enables you to use this command asynchronously, such as in a batch file that is distributed to a computer where a user is not logged in. The command is actually executed when a user logs in to the computer. You must use the command in a batch file if you specify this option. This option is supported on Windows operating systems.

- `–c "command_line"`
  Specifies the command line invoked by the icon.

- `–g group_name`
  Specifies the program group name to which the icon is added.

- `–i icon_file`
  Specifies the file containing the icon. If this option is not specified, the Program Manager looks in the executable file specified by the `–c` option.

- `–m "message"`
  Specifies a message to be written to the waddicon.err file if an error occurs when using this command with the `–a` option. This option is supported on Windows operating systems.

- `–r`
  Removes the specified icon. If no icon is specified, attempts to remove the entire program group.

- `–t icon_title`
  Specifies the title (description) below the icon in the Program Manager.

Return Values

Returns one of the following:

- `0` Indicates the successful addition of the icon.
- non-zero Indicates an unsuccessful addition of the icon.

Examples

1. The following example adds the **Word Processor** icon for the word processor application to the **Applications** Program Manager group:

```
waddicon -g "Applications" -c \W\PROCESS.EXE -t "Word Processor"
```
2. The following example removes the **Word Processor** icon from the **Applications** group:
   ```
waddicon -g "Applications" -c \WP\WPRESS.EXE -t "Word Processor" -r
   ```

3. The following example removes the **Applications** group:
   ```
waddicon -g "Applications" -r
   ```

4. The following example adds an icon to a user desktop (on a Windows workstation), includes the following command in a batch file and distributes the file to the computer. If an error occurs, the message following the **-m** option is written to the `WADDICON.ERR` file.
   ```
c:\tivoli\tmeagent\win32\cli\waddicon -c "C:\FILES\MY_PROG.EXE" \t"My Program" -m "Call #1 of waddicon" -a
   ```
waddpath

Adds an entry to the path statement in the registry hive of the current control set. This command should be run from an endpoint. (Windows 2000 only)

**Syntax**

```
   waddpath path_value
```

**Description**

The `waddpath` command adds an entry to the Windows \\SYSTEM\\CurrentControlSet\\Control\\SessionManager\\Environment key path in the HKEY_LOCAL_MACHINE registry hive. After you add the entry, other applications have a search path to the application.

This command returns a message indicating if it successfully completed or encountered an error.

**Options**

```
   path_value
```

Specifies the path entry to add to the registry hive in the current control set.

**Authorization**

```
   admin
```

**Examples**

To add the `\APP\MISC\EXEC` directory path to the registry hive, enter the following command:

```
waddpath \APP\MISC\EXEC
```
wadminep

Performs administrative operations on an endpoint.

Syntax

wadminep –h

wadminep endpoint_name wake_up

Description

The wadminep command is run from a managed node but administers all supported endpoint platforms. Use this command to generate the endpoint wake-up packet for wake-on-LAN operations. To enable wake-on-LAN support, use the –w option on the lcfd command. By default, wake-on-LAN is not enabled on endpoints.

Options

–h Displays a detailed usage statement listing all available command options.

wake_up Broadcasts a wake-up packet over the network to the specified endpoint.

endpoint_name Specifies the name of the endpoint on which the command runs.

Authorization

The super role is required with the privilege of root or Administrator on the endpoint. The role can be resource-specific only if the resource is a policy region and the endpoint belongs to that policy region.

Examples

The following example generates the endpoint wake-up packet for the endpoint pine:

wadminep pine wake_up

See Also

lcfd, winstlcf
**wauthadmin**

Adds, revokes, or displays the Tivoli root administrators in a Tivoli region.

**Syntax**

```plaintext
wauthadmin –a administrator

wauthadmin –r administrator

wauthadmin –l [–v]
```

**Description**

The `wauthadmin` command serves two purposes in the management of Tivoli Administrator accounts:

- Lists all root administrators in a Tivoli region.
- Allows an existing root administrator to grant and revoke root authorization.

When you install a Tivoli server, the process creates an initial Tivoli account called the root administrator. This account possesses full, unrestricted privileges for accessing and managing systems. On UNIX operating systems, the Tivoli root administrator account has UNIX root account permissions; on Windows operating systems, the account has Administrator account permissions. The `wauthadmin` command enables a root administrator to promote existing Tivoli administrators to root administrators.

The administrator to be promoted must have at least one global authorization role in the local Tivoli region. If all global roles are removed from a root administrator, root authorization for this administrator is revoked.

The `wauthadmin` command also removes root authority from root administrator accounts. You must, however, retain at least one root administrator for each Tivoli region.

With the `–l` option, the command displays a list of Tivoli administrators who possess root authority in the local Tivoli region.

**Options**

- `–a administrator`  
  Adds root authorization to the specified administrator.
- `–l`  
  Lists administrators who are root administrators in the local Tivoli region.
- `–r administrator`  
  Revokes root authorization from the specified administrator.
- `–v`  
  Provides verbose information about the listed root administrators. This option can only be used with the `–l` option.

**Authorization**

Only a root administrator can grant or revoke root authority, but requires user to display the list of root administrators.
Examples

1. The following example grants root authority to administrator kimiko@ohio in the local Tivoli region:
   wauthadmin -a kimiko@ohio

2. The following example displays a verbose, detailed list of root administrators in the local Tivoli region:
   wauthadmin -l -v

See Also

wgetadmin  wsetadmin
wbkupdb

Backs up and restores Tivoli object databases.

Syntax

```
wbkupdb -e [node_name...]
wbkupdb -e -l [object node_name...]
wbkupdb [-b] [-d device] [-f] [-h node_name] [node_name...]
wbkupdb -l [-d device] [-f] [-h node_name] [object node_name...]
wbkupdb -d device -r [-R] [node_name...]
wbkupdb -d device -r -l [-R] [object node_name...]
wbkupdb -s
```

Description

The `wbkupdb` command backs up and restores Tivoli object databases. You can provide a list of managed node names as options to the `wbkupdb` command. If you do not specify any node options, the `wbkupdb` command backs up or restores the Tivoli object database for every managed node in the Tivoli region.

Note: If you are unable to capture a full backup because of the high activity level of your Tivoli region, you can use the `wlocktmr` command to place your Tivoli region in maintenance mode and then run the `wbkupdb` command again. For more information about placing a Tivoli region in maintenance mode, refer to Tivoli Management Framework Maintenance and Troubleshooting Guide.

When the `-e` option is used, this command estimates the total size of the backup archive. The `wbkupdb -e` command estimates the size of the backup of each managed node and the total size of the archive. This report is an estimate, but it is very close to the actual size of the backup image.

The third and fourth forms of this command (as shown in the syntax) back up the database and store it in the specified file or device on the specified system. If the backup file already exists and is a disk file, you must specify the `-f` option to overwrite the old backup file.

When the `-r` option is used, this command restores Tivoli object databases. This is primarily useful for reverting to a previously saved copy of the Tivoli object databases that you backed up. The Tivoli server or managed node that is to be restored must have Tivoli Management Framework operational. If a restore operation is being performed from a system other than the Tivoli server, you cannot restore both the Tivoli server and the local database unless you specify the `-R` option. If you restore the local database, you must use the explicit `node_name` syntax and specify the local node at the end or specify the `-R` option.
If the object dispatcher that is to be restored is not running (and presumably cannot be run because its database is corrupted or missing), you can extract the database manually and put the files in the correct location in the database directory.

The `wbkupdb` command also saves any old versions of files and the notification database. Typically, these are not restored, because you probably do not want to read notices that have already been read. If for some reason the file is destroyed, you can restore it manually. The `\files_versions` directory is not restored. If you want to see old revisions of system files, the files can be moved from the `\files_versions.restore` directory as necessary.

The following list details the files and databases backed up in the temporary directory on UNIX and Windows managed nodes and Tivoli servers:

- For UNIX managed nodes, recovered files are as follows:
  - `odb.bdb.restore`
  - `odb.adj.restore`
  - `odb.log.restore`
  - `\file_versions.restore` (directory)

- For a UNIX Tivoli server, recovered files are as follows:
  - `odb.bdb.restore`
  - `odb.log.restore`
  - `imdb.bdb.restore`
  - `notice.bdb.restore`
  - `notice.log.restore`
  - `odlist.dat.restore`
  - `\epmgr.bdb.restore` (directory)
  - `\file_versions.restore` (directory)

- For Windows managed nodes, recovered files are as follows:
  - `odb.bdb`
  - `odb.adj`
  - `odb.log`
  - `\file_versions` (directory)

- For a Windows Tivoli server, recovered files are as follows:
  - `odb.bdb`
  - `odb.log`
  - `imdb.bdb`
  - `notice.log`
  - `odlist.dat`
  - `\epmgr.db` (directory)
  - `\files_versions` (directory)

**Options**

`-b` Blocks the writing of data to the Tivoli object databases during the backup. When you specify this option, the object dispatcher does not write data to the Tivoli object database while the database is backed up. After the backup operation completes, any write operations that queued during the backup operation are written to the database. Use the `-b` option when you
want to ensure that the Tivoli object databases are unchanged for the
duration of the backup, for example, in a very busy Tivoli environment. If
you use the -b option to back up a node that has not been upgraded to a
version of Tivoli Management Framework that contains this feature, the
standard backup method is used for that node.

The -b option causes the backup operation to check for needed disk space
before performing the backup. A warning message is displayed if sufficient
disk space is not available.

Note: The needed disk space is estimated by calculating twice the sum of
the sizes of odb.bdb and imdb.bdb. In some environments, this
estimate might be too low. For these situations, you can set the
TIVBACKUPSIZE environment variable to override the estimate
produced by the -b option. Set the value for TIVBACKUPSIZE in
KB. On Windows operating systems, TIVBACKUPSIZE must be a
system environment variable.

When you specify this option, the object dispatcher writes error messages
that result from the backup to the oservlog file.

`-d device`
Specifies the file or device to which the backup file should be saved or
from which the backup file should be retrieved. If you specify a file name,
you can insert a file date and time anywhere in the file name by adding
the %t variable. This variable is replaced with a time stamp in the form
`Mondd-hmmn`. For example, if you specify `-d /usr/backups/TMR1%t.bk`, the
resulting file is named TMR1Dec21-0955.bk. The time is displayed in
24-hour mode.

`-e` Estimates the size of the backup.

`-f` Overwrites a previous backup file of the same name.

`-h node_name`
Specifies the system that contains the .tar file that is created by the
`wbkupdb` command. The default is the Tivoli server.

`-i` Specifies that options on the command line are object and label pairs. This
option is for internal use only.

`-r` Restores the databases for the specified nodes.

`-R` Does not restart object dispatchers after restoring the database files. A
number of .restore files are placed in the database directory. To effect
the changes, use the reexec option of the odadmin command or one of its
derivatives to restart the object dispatchers so that they pick up the
restored copies of the databases.

`-s` Suppresses display of an initial desktop message for each managed node
being backed up.

`node_name`
Specifies the node to be backed up. You can specify multiple nodes.

`object` Specifies a backup object identifier.

**Authorization**
To create a backup, the `backup` role in the Tivoli region is required.
To restore from a backup, the **restore** role in the Tivoli region to perform a restore operation is required.

The default backup directory requires root write permission. Log in as the root administrator or change the location of the backup file. If you are performing a “rescue” operation, you must be root on the machine where the crashed database is located.

**Examples**

1. The following example backs up the Tivoli object databases for all managed nodes in the Tivoli region, and writes the backups to the /usr/backups/TMR1.bkfile:
   ```
   wbkupdb -d /usr/backups/TMR1.bk
   ```
2. The following example backs up the database of managed node sherman. Because no backup file is specified, the backup is written to the $DBDIR/backups directory.
   ```
   wbkupdb sherman
   ```
3. The following example restores the database to managed node sherman by using the /usr/backups/TMR1.bk backup file:
   ```
   wbkupdb -r -d /usr/backups/TMR1.bk sherman
   ```

**See Also**

-wchkdb, -wclient, -winstall, -wpatch, -wserver
wbindmsg

Retrieves a translated string from a local message catalog and binds any variables.

**Syntax**

```
wbindmsg catalog_name message_number default_string [options...]
```

**Description**

The `wbindmsg` command retrieves the message corresponding to specified message number from the specified message catalog. If the specified message cannot be retrieved in the current language environment, then the `default_string` is used. The `wbindmsg` always looks for the specified message in set 2 of the specified message catalog.

The `wbindmsg` command uses the LANG and NLSPATH environment variables to find the message catalog appropriate for the current language environment. For example, if LANG is equal to `fr_FR` and NLSPATH is equal to `/tivoli/msg_cat/%L/%N.cat;/tivoli/msg_cat/%L%C/%N.cat` the `wbindmsg` command attempts to find the message catalog by using the following path names:

- `/tivoli/msg_cat/fr_FR/catalog_name.cat`
- `/tivoli/msg_cat/fr/catalog_name.cat`
- `/tivoli/msg_cat/C/catalog_name.cat`

After the message is retrieved and bound, the resulting string is written to standard output.

**Options**

- `catalog_name`
  Specifies the message catalog base name.

- `message_number`
  Specifies the message number in the message catalog.

- `default_string`
  Specifies the string to used if the message cannot be retrieved from the message catalog.

- `[options...]`
  Specifies any options to be bound in place of the format directives in the message.

**Examples**

The following example retrieves the second message from the `my_catalog.cat` message catalog. The default string is the same as the English version of the message, which contains two options, jross and polyglot.

```
wbindmsg my_catalog 2 "User %1$s does not have an account on %2$s" jross polyglot
```

User jross does not have an account on polyglot.
wbroadcast

Broadcasts a message to all Tivoli desktops.

Syntax

```
wbroadcast [optional_text]
```

Description

The `wbroadcast` command reads a message from standard input and broadcasts it to each Tivoli desktop in the Tivoli environment. If the `optional_text` option is used, this text, rather than standard input, is broadcast to all the desktops.

Options

```
optional_text
```

Specifies the text to be broadcast.

Authorization

admin, senior, super

Examples

The following example is a typical broadcast message:

```
wbroadcast
<< EOF
Restoring database in 5 minutes. Exit your desktop.
EOF
```
**wcatcher**

Saves custom dialogs before upgrading a Tivoli Enterprise application.

**Syntax**

```
wcatcher -a [-d parent_dir] [-s sub_dir] [-v]
wcatcher -r resource... [-d parent_dir] [-s sub_dir] [-v]
```

**Description**

The **wcather** command saves custom dialogs for a Tivoli Enterprise application before upgrading. Examples of custom dialogs are those with added text fields or buttons.

For example, if you are preparing to upgrade to a new version of a Tivoli Enterprise application, use this command to save your custom dialogs. After upgrading, use the **wmrgaef** command to reapply the custom dialogs to the new version.

The **wcather** command stores each custom dialog, and its corresponding original dialog, in a directory structure that you specify with the **-d** option. The **wcather** command creates the custom.sav directory unless you specify a different name with the **-s** option, and then searches each specified resource type to find custom dialogs. For each dialog, it saves the custom version and the original version in a subdirectory under the custom.sav directory that has the same name as the resource. If a dialog has been customized on a per-instance basis, the dialogs are saved in a directory that has the name of the label instance.

**Options**

- **-a** Searches all registered resources for custom dialogs.
- **-d parent_dir**
  Specifies the path to the parent directory where each custom dialog and its corresponding original version is saved. If you do not supply a path, the command writes the information to the current working directory.
- **-r resource**
  Searches a specific resource type. You can specify multiple resource types on a command line. For each resource type, you must supply **-r**.
- **-s sub_dir**
  Specifies the name of the directory where where each custom dialog and its corresponding original version is saved. If yo do not supply a directory name, the command creates the custom.sav directory.
- **-v** Specifies verbose output.

**Authorization**

super

**Examples**

The following command searches for all custom dialogs and saves the files to the /tmp/aef/my.dir directory:

```
wcather -d /tmp/aef -s my.dir
```
See Also

wmrgaeft
**wcd**

Changes the current working collection.

**Syntax**

```
wcd [label]
```

**Description**

The `wcd` command changes the current working collection object to the collection object specified by the given label path. Each administrator has a separate current working collection associated with each parent process ID.

**Options**

- `label`: Specifies the new working collection; the specified object must be a collection object or policy region. This option can be a full label path (starting at the `/collection`) or a partial label path (relative to the current working collection). If this option is omitted, this command changes the working collection to the home collection. The home collection for each administrator `/Administrators/name`, where `name` is the principal name of the administrator.

**Authorization**

`user, admin, senior, super`

**Examples**

1. The following example changes the current working collection to the ceridwen-region policy region:
   ```
wcd /Regions/ceridwen-region
   ```
2. The following example changes the current working collection to the object database directory:
   ```
wcd $DBDIR
   ```

**See Also**

`wls` `wpwd`
wchdep

Associates a dependency set with a method header.

Syntax

wchdep @Classes:class_name dependency_set method_name ...

wchdep –g @Classes:class_name method_name

Description

The wchdep command associates a dependency set with a particular method header. You must create the dependency set using the –c option of the wdepset command before using the wchdep command to specify which method header to associate the dependency set.

You can only associate one dependency set with each method, but you can have nested dependency sets as well as use the –e option of the wdepset command to add to an existing dependency set.

The –g option of the wchdep command returns the object ID (OID) of the dependency set. The OID can be used with the wdepset command so you do not need to look up the label of the dependency set.

The wchdep command is usually placed in the object database after script or in an .ist file of an application.

Options

–g Returns the OID of the dependency set.
@Classes:class_name The name of the dependency class object.

dependency_set The label associated with the wdepset command. You can specify a dependency set as a label in one format @DependencyMgr:label or as an OID.

method_name The name of the method the dependency method supports.

Authorization

user

Examples

The following example associates the hello_dependency dependency set with the hello_method method within the Hello dependency class object.

wchdep @Classes:Hello hello_dependency hello_method

See Also

wdepset
wchkdb

Verifies and repairs the Tivoli database.

Syntax

wchkdb [-o outfile] [-t] [-u] [-x] -f infile
wchkdb [-o outfile] [-t] [-u] [-x] -i
wchkdb [-o outfile] [-t] [-u] [-x] object...

Description

The wchkdb command verifies and repairs problems in the Tivoli database. This command does not affect any system files; it only modifies resources within the Tivoli region.

Without the -u option, wchkdb only verifies the database and reports discrepancies to standard output or, optionally, to an output file if the -o option is used. The output file can later be passed to wchkdb with the -u and -f options to correct the discrepancies.

The -u option attempts to find and fix any database discrepancies. The -f option reads input from an output file generated in a previous run of wchkdb. All objects in the infile are checked. The -i option reads a list of objects from standard input. You can also specify a list of objects on the command line. If no input options or object references are provided, all objects in the database are checked.

The -x option enables verification across Tivoli region boundaries. Without this option, only resources within the current region are verified and repaired.

In a high-activity Tivoli environment, it can be advantageous to place a Tivoli management region in maintenance mode, thereby stopping all active Tivoli processes. For more information about placing a Tivoli region in maintenance mode, see Tivoli Management Framework Maintenance and Troubleshooting Guide.

Options

- -f infile
  Reads the specified input file, which was created during a previous database check, and checks only those objects listed in the file. Only objects that failed during the previous run are included in the file.

- -i
  Reads a list of objects from standard input. The list consists of object IDs, object names, or both (each separated by a space).

- -o outfile
  Writes a binary version of the displayed output to the specified file name.

- -t
  Verifies and repairs object references in only the Tivoli server database.

- -u
  Updates the database, fixing any discrepancies found in the Tivoli resource database.

- -x
  Verifies and repairs object references across region boundaries.

object...

Specifies an object ID or object name to be verified and repaired.
Authorization

senior or super

Return Values

0 Indicates that the operation started successfully.
nonzero Indicates that the operation did not start successfully, either because of a syntax error or because the object dispatcher is not available.

Examples

1. The following example verifies and, if needed, repairs the Tivoli database. Object references are checked across Tivoli region boundaries.
   wchkdb -u -x

2. The following example verifies object references in the current Tivoli region only. No changes are made to the Tivoli database. Problems, however, are displayed to standard output and written to the /tmp/check.out binary output file.
   wchkdb -o /tmp/check.out

3. The following example reads the results from a previous run of wchkdb invocation (/tmp/check.out) and updates the Tivoli database as needed:
   wchkdb -u -f /tmp/check.out
wchknode

Verifies and updates references to a specific dispatcher number from parts of the Tivoli database.

**Syntax**

```
wchknode [-c] [-n] [-s] [-u] [-v] [-x] [dispatcher_num]
```

**Description**

The `wchknode` command synchronizes the object database on the Tivoli server with the database of managed node specified by its object dispatcher number. The command verifies the Tivoli name registry with the `-n` option, Tivoli collections with the `-c` option, or subscription lists with the `-s` option. If none of these options are specified, the `wchknode` command checks all these databases.

When the `-u` option is specified, the `wchknode` command updates all references to `dispatcher_num`. When omitted, `wchknode` finds all references and prints discrepancies to the screen. When no dispatcher number is specified, the `wchknode` command verifies all managed nodes the Tivoli region.

The `wchknode` command is run from the Tivoli server. By default, the command runs only in the local Tivoli region. To verify resources in connected Tivoli regions, use the `-x` option.

When you do not have time to run a full `wchkdb`, run the `wchknode` command to check the references to a dispatcher number after a managed node or dispatcher database is restored.

**Note:** Invoke the `wchknode` command only on managed nodes that are in a connected and communicating state. If you invoke this command on a managed node that is down, the managed node is deleted from the Tivoli database. Also, do not invoke this command without specifying the target dispatcher number of a managed node. If you issue this command without specifying a target, any managed node with a downed oserv is deleted from the Tivoli database.

**Options**

- `-c` Verifies references to `dispatcher_num` in Tivoli collections.
- `-n` Verifies references to `dispatcher_num` in the Tivoli name registry.
- `-s` Verifies references to `dispatcher_num` in subscription lists.
- `-u` Updates any discrepancies found during the verification. When omitted, the command prints all discrepancies to the screen.
- `-v` Prints verbose messages to the screen.
- `-x` Verifies references to `dispatcher_num` across Tivoli region boundaries.

`dispatcher_num` Identifies the dispatcher number of the managed node to verify. To determine the correct dispatcher number, use the `odadmin odlist` command.
Authorization

**senior** or **super** in the Tivoli region

Examples

1. The following example verifies and updates all references to managed node yellow (dispatcher 7) in the Tivoli name registry, Tivoli collections, and subscription lists. This operation searches the local Tivoli region and all connected Tivoli regions.
   \[ \text{wckhnode -u -x 7} \]

2. The following example verifies the Tivoli name registry for references to managed node thor (dispatcher 5). References are not removed and are printed to the screen.
   \[ \text{wckhnode -n 5} \]

See Also

[**wchkdb**] [**wrmnode**]
wchkpol

Verifies policy region members against policy.

Syntax

```
wchkpol -a [-c collection] [-f file] policy_region_name
wchkpol [-c collection] [-f file] -l label policy_region_name
wchkpol [-c collection] [-f file] -r resource_type policy_region_name
```

Description

The `wchkpol` command verifies that all or a subset of the members of a policy region comply with the policy enabled for the region. For each member that does not pass the policy verification, a message is sent to standard output. The `-f` option sends the output to a log file. The `-c` option, can be used to create a collection of those members that fail the policy verification.

Options

- `a`  
  Verifies all members of the policy region.

- `c collection`
  
  Creates a collection for members that failed the policy verification.

- `f file`
  
  Creates a log file containing a list of members that failed the policy check.

- `l label`
  
  Verifies the resource specified by `label`.

- `r resource_type`
  
  Verifies all policy region members that are of the resource type specified.

- `policy_region_name`
  
  Specifies the policy region whose members are to be verified.

Authorization

admin, senior, super

Examples

1. The following example verifies the policy for all members of the ceridwen-region policy region:
   ```
   wchkpol -a /Regions/ceridwen-region
   ```

2. The following example verifies the policy for all managed nodes in the ceridwen-region policy region and creates the failures collection to contain any managed node that does not pass the policy verification:
   ```
   wchkpol -c failures -r ManagedNode ceridwen-region
   ```

See Also

`wcrtpol`, `wcrtpr`, `wdelpr`, `wgetdfpol`, `wgetpolm`, `wlspol`, `wlspolm`, `wputpolm`
wci

Checks in Revision Control System (RCS) revisions.

**Syntax**

```
wc [options] file...
```

**Description**

The *wci* command stores new revisions into RCS files. Each path name matching an RCS suffix is taken to be an RCS file. All others are assumed to be working files containing new revisions. The command deposits the contents of each working file into the corresponding RCS file. If only a working file is given, the command tries to find the corresponding RCS file in an RCS subdirectory and then in the working file directory. For more details, see “File Naming” on page 101.

For the *wci* command to work, the caller login must be on the access list, unless the access list is empty or the caller is the super user or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file if non-strict locking is used (see the *wrcs* command). A lock held by someone else can be broken with the *wrcs* command.

Unless the `-f` option is given, the *wci* command checks whether the revision to be deposited differs from the preceding one. If not, instead of creating a new revision, the command reverts to the preceding one. To revert:

- Use the *wci* command without options removes the working file and any lock.
- Use the *wci* command with the `-I` keeps locks and generates a new working file as if using the *wco* command on the preceding revision.
- Use the *wci* command with the `-u` removes locks and generates a new working file as if using the *wco* command on the preceding revision.

When reverting, any `-n` and `-s` options apply to the preceding revision.

For each revision deposited, the *wci* command prompts for a log message. The log message should summarize the change and must be terminated by end-of-file or by a line containing a period (.) by itself. If several files are checked in, the command prompts whether to reuse the previous log message. If the standard input is not a terminal, the command suppresses the prompt and uses the same log message for all files (see `-m`).

If the RCS file does not exist, the *wci* command creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, the command requests descriptive text (see `-t`).

The number *rev* of the deposited revision can be given by any of the options `-f`, `-I`, `-k`, `-l`, `-M`, `-q`, `-r`, or `-u`. The *rev* option can be symbolic, numeric, or mixed:

- If *rev* is $, the command determines the revision number from keyword values in the working file.
- If *rev* is a revision number, it must be higher than the latest one on the branch to which *rev* belongs or must start a new branch.
If `rev` is a branch rather than a revision number, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If `rev` indicates a nonexisting branch, that branch is created with the initial revision numbered `rev.1`.

If `rev` is omitted, the `wci` command tries to derive the new revision number from the last lock of the caller. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1.

If `rev` is omitted and the caller has no lock but owns the file and locking is not set to `strict`, the revision is appended to the default branch (typically the trunk; see the `-b` option of the `wrcs` command).

The following exception applies on the trunk, revisions can be appended to the end, but not inserted.

An RCS file created by the `wci` command inherits the read and execute permissions from the working file. If the RCS file exists, the command preserves its read and execute permissions. The command always turns off all write permissions of RCS files.

**Options**

```
-l [rev]
Works like -r, except it performs an additional `wco -l` for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although you want to continue editing it after the check-in.

-r [rev]
Checks in a revision, releases the corresponding lock, and removes the working file. This is the default.

The -r option has an unusual meaning in `wci`. In other RCS commands, -r merely specifies a revision number, but in `wci` it also releases a lock and removes the working file. See -u for an example.

-u [rev]
Works like -l, except that the deposited revision is not locked. This lets you read the working file immediately after check-in.
```

The -l, -r, and -u options are mutually exclusive and silently override each other. For example, `wci -u -r` is equivalent to `wci -r` because the -r option overrides the -u option.

```
-d "[date]"
Uses `date` for the check-in date and time. The date is specified in free format as explained in the `wco` command. This is useful for specifying a check-in date other than the actual date, and for -k if no date is available. If the date is omitted, the working file time of last modification is used.

-f [rev]
Forces a deposit; the new revision is deposited even if it is not different from the preceding one.

-k [rev]
Searches the working file for keyword values to determine its revision number, creation date, state, and author (see `wco`), and assigns these values
to the deposited revision, rather than computing them locally. It also generates a default login message noting the login of the caller and the actual check-in date. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the \texttt{-k} option at these sites to preserve the original number, date, author, and state. The extracted keyword values and the default log message may be overridden with the options \texttt{-d, -m, -s, -w}, and any option that carries a revision number.

\texttt{-I \[rev\]} Interactive mode; the user is prompted and questioned even if the standard input is not a terminal.

\texttt{-m \[message\]} Uses the string \texttt{message} as the log message for all revisions checked in.

\texttt{-M \[rev\]} Sets the modification time on any new working file to be the date of the retrieved revision. For example, \texttt{wci -d -M -u file} does not alter the modification time for \texttt{file}, even when its contents change because of keyword substitution. Use this option carefully; it can confuse the \texttt{make} command.

\texttt{-n \[name\]} Assigns the symbolic name \texttt{name} to the number of the checked-in revision. The \texttt{wci} command prints an error message if \texttt{name} is already assigned to another number.

\texttt{-N \[name\]} Same as \texttt{-n}, except that it overrides a previous assignment of \texttt{name}.

\texttt{-q \[rev\]} Quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless \texttt{-f} is given.

\texttt{-s \[state\]} Sets the state of the checked-in revision to the identifier \texttt{state}. The default state is \texttt{Exp}.

\texttt{-t \[file\]} Writes descriptive text from the contents of \texttt{file} into the RCS file, deleting the existing text. The \texttt{file} may not begin with a dash (\texttt{-}).

\texttt{-t \[string\]} Writes descriptive text from the \texttt{string} into the RCS file, deleting the existing text.

The \texttt{-t} option, in both its forms, has effect only during an initial check-in; it is silently ignored otherwise.

During the initial check-in, if \texttt{-t} is not given, the \texttt{wci} command obtains the text from standard input, terminated by end-of-file or by a line containing a period (\texttt{.}) by itself. The user is prompted for the text if interaction is possible; see \texttt{-I}.

For backward compatibility with older versions of RCS, \texttt{-t} without \texttt{string} is ignored.

\texttt{-V \[n\]} Emulates RCS version \texttt{n}. See the \texttt{wco} command for details.

\texttt{-w \[login\]} Uses \texttt{login} for the author field of the deposited revision. Useful for specifying an author other than the actual author, and for \texttt{-k} if no author is available.
-x suffixes

Specifies the suffixes for RCS files. A nonempty suffix matches any path name ending in the suffix. An empty suffix matches any path name of the form RCS/file or path/RCS/file. The -x option can specify a list of suffixes separated by a slash (/). For example, -x,v/a specifies two suffixes: v and the empty suffix. If two or more suffixes are specified, they are tried in order when looking for an RCS file; the first one that works is used for that file. If no RCS file is found but an RCS file can be created, the suffixes are tried in order to determine the new RCS file name. The default for suffixes is installation-dependent; typically it is ,v/ for hosts such as UNIX hosts that permit commas in file names and is empty (for example, just the empty suffix) for other hosts.

Using SETUID on UNIX

To prevent anybody but their RCS administrator from deleting revisions, a set of users can employ setuid privileges as follows:

- Check that the host supports RCS setuid use. Consult a trustworthy expert if there are any doubts. It is best if the setuid() system call works as described in Posix 1003.1a Draft 5, because RCS can switch back and forth easily between real and effective users, even if the real user is root. If not, the second best use is if the setuid() system call supports saved setuid (the [POSIX_SAVED_IDS] behavior of Posix 1003.1-1990); this fails only if the real user is root. If RCS detects any failure in setuid, it quits immediately.
- Choose a user Ali to serve as RCS administrator for the set of users. Only Ali is able to invoke the wrcs command on the RCS files of the users. Ali should not be root or any other user with special powers. Mutually suspicious sets of users should use different administrators.
- Choose a path name bronte that is a directory of files to be executed by the users.
- Have Ali set up bronte to contain copies of the wci and wco command that are setuid to Ali by copying the commands from their standard installation directory D as follows:
  
  ```
  mkdir bronte
  cp b/wc[io] bronte
  chmod go-w,u+s bronte/wc[io]
  ```
- Have each user prepend bronte to their path as follows:
  
  ```
  PATH=bronte:$PATH; export PATH # ordinary shell
  set path=(bronte $path) # C shell
  ```
- Have Ali create each RCS directory rambault with write access only to Ali as follows:
  
  ```
  mkdir rambault
  chmod go-w rambault
  ```
- If you want to let only certain users read the RCS files, put the users into the gaming group, and have Ali further protect the RCS directory as follows:
  
  ```
  chgrp gaming rambault
  chmod g-w,o-rwx rambault
  ```
- Have Ali copy old RCS files (if any) into rambault, to ensure that Ali owns them.
- The access list for an RCS file limits who can check in and lock revisions. The default access list is empty, which grants check-in access to anyone who can read the RCS file. If you want to limit check-in access, have Ali run wrcs -a Ali on the file; see the wrcs command. For example, wrcs -e -a Ali limits access to just Ali.
- Have Ali initialize any new RCS files with `wrcs -i` before initial check-in, adding the `-a` option if you want to limit check-in access.
- Give setuid privileges only to the `wci` and `wco` commands.
- Do not use other setuid commands to invoke RCS commands.

**Diagnostics**

For each revision, the `wci` command prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status is zero if and only if all operations were successful.

**Files**

Several temporary files might be created in the directory containing the working file and also in the temporary directory (see `TMPDIR` in the section "Environment Variables" on page 102). Semaphore files are created in the directory containing the RCS file. With a nonempty suffix, the semaphore names begin with the first character of the suffix; therefore, do not specify a suffix whose first character could be that of a working file name. With an empty suffix, the semaphore names end with an underscore (_), so working file names should not end with an underscore.

The `wci` command never changes an RCS or working file. Typically, the command unlinks the file and creates a new one; but instead of breaking a chain of one or more symbolic links to an RCS file, it unlinks the destination file instead. Therefore, the command breaks any hard or symbolic links to any working file it changes; and hard links to RCS files are ineffective, but symbolic links to RCS files are preserved.

The effective user must be able to search and write the directory containing the RCS file. Typically, the real user must be able to read the RCS and working files and to search and write the directory containing the working file; however, some older hosts cannot easily switch between real and effective users, so on these hosts the effective user is used for all accesses. The effective user is the same as the real user unless your copies of `wci` and `wco` have setuid privileges. As described in the next section, these privileges yield extra security if the effective user owns all RCS files and directories and if only the effective user can write RCS directories.

Users can control access to RCS files by setting the permissions of the directory containing the files; only users with write access to the directory can use RCS commands to change its RCS files. For example, in hosts that allow a user to belong to several groups, one can make a group RCS directories writable to that group only. This approach suffices for informal projects, but it means that any group member can arbitrarily change the group RCS files and can even remove them entirely. Hence more formal projects sometimes distinguish between an RCS administrator, who can change the RCS files at will, and other project members, who can check in new revisions but cannot otherwise change the RCS files.

**File Naming**

Pairs of RCS files and working files can be specified in three ways:
- Both the RCS file and the working file are given. The RCS path name is of the form `path1/workfileX` and the working path name is of the form `path2/workfile` where `path1` and `path2` are (possibly different or empty) paths, `workfile` is a file name, and `X` is an RCS suffix. If `X` is empty, `path1` must be RCS/ or must end in `/RCS/.

Chapter 3. Commands alphabetically 101
• Only the RCS file is given. Then the working file is created in the current
directory and its name is derived from the name of the RCS file by removing
\textit{path1} and the suffix \textit{X}.
• Only the working file is given. Then the \texttt{wci} command considers each RCS
suffix \textit{X} in turn, looking for an RCS file of the form \texttt{path2/RCS/workfileXa}
or (if the former is not found and \textit{X} is nonempty) \texttt{path2/workfileX}.

If the RCS file is specified without a path in methods 1 and 2, the \texttt{wci} command
looks for the RCS file first in the ./RCS directory and then in the current directory.

The \texttt{wci} command reports an error if an attempt to open an RCS file fails for an
unusual reason, even if the RCS file path name is just one of several possibilities.
For example, to suppress the use of RCS commands in directory dave, create a
regular file named dave/RCS so that casual attempts to use RCS commands in
dave fail because dave/RCS is not a directory.

\section*{Environment Variables}

\textbf{RCSINIT}

Options attached to the option list, separated by spaces. A backward slash
escapes spaces within an option. The \texttt{RCSINIT} options are prepended to
the option lists of most RCS commands. Useful \texttt{RCSINIT} options include
\texttt{–q}, \texttt{–V}, and \texttt{–x}.

\textbf{TMPDIR}

Name of the temporary directory. If not set, the environment variables
\texttt{TMP} and \texttt{TEMP} are inspected instead and the first value found is taken; if
none of them are set, a host-dependent default is used, typically the \texttt{/tmp}
directory.

\section*{Examples}

1. Suppose \texttt{,v} is an RCS suffix and the current directory contains a subdirectory
RCS with an RCS file \texttt{io.c,v}. Then each of the following commands checks a
copy of \texttt{io.ca} into RCS/\texttt{io.c,v} as the latest revision, removing \texttt{io.c}:
\begin{verbatim}
 wci io.c; wci RCS/\texttt{io.c,v}; wci \texttt{io.c,v};
 wci \texttt{io.c} RCS/\texttt{io.c,v}; wci \texttt{io.c} \texttt{io.c,v};
 wci RCS/\texttt{io.c,v} \texttt{io.c}; wci \texttt{io.c,v} \texttt{io.c};
\end{verbatim}

2. Suppose instead that the empty suffix is an RCS suffix and the current
directory contains a subdirectory RCS with an RCS file \texttt{io.c}. Each of the
following commands checks in a new revision:
\begin{verbatim}
 wci \texttt{io.c}; wci RCS/\texttt{io.c};
 wci \texttt{io.c} RCS/\texttt{io.c};
 wci RCS/\texttt{io.c} \texttt{io.c};
\end{verbatim}

\section*{Author}

Author: Walter F. Tichy. Revision Number: 5.9; Release Date: 1991/110/07.
Eggert.

\section*{See Also}

\texttt{wci} \texttt{wident} \texttt{wrsc} \texttt{wrscdiff} \texttt{wrscmerge} \texttt{wrlog}\texttt{...} Walter F. Tichy, RCS—A System
**wclient**

Creates a managed node.

**Syntax**

```
```

**Description**

The `wclient` command creates a managed node when invoked on a managed node.

**Options**

- **-c source_dir**
  Specifies the complete path to the directory containing the installation image. For `source_dir`, you can specify either the fully qualified path (for example `/cdrom`) or the source host and the path in the format `source_host:path` (for example `oak:/cdrom`).

- **-d**
  Sets `install_variables` to their last set values. Commonly, each installed managed node uses the same set of installation variables. This option provides a shortcut for setting the variables.

- **-f file_name**
  Specifies a text file containing a list of machines to be installed. The file contains one line per machine with the format `host_name,user,password`, where `user` and `password` are optional. If not specified, the default user is root or Administrator. There can be no spaces between entries. The content of each line determines the default method.
  - For the default, each entry contains only the machine name.
  - For account access, each entry contains the machine name followed by a comma (,), the user ID followed by a comma (,), and then the password.

  **Note:** Passwords are not encrypted. Anyone with access to this file can see the passwords.
  - For trusted host access, each entry contains the machine name followed by a comma (,).

  The following is an example of a machine file:

  ```
elm,
oak,chris,rews
liveoak,
```

- **-I**
  Causes the command to prompt for the installation password. If you do not use this option, there is not a password prompt.

- **-j**
  Causes the command to use an encrypted secure shell (SSH) connection when the Tivoli server connects to the machine to create the managed node.

  **Note:** You cannot use SSH to perform installations on Windows targets.

- **-p policy_region**
  Specifies the name of the policy region in which the managed nodes are installed.
Causes the command to prompt for the root password for the machine. If you specify more than one machine, the command assumes they all have the same root password.

Specifies that the account and password specified with the -U option should be used as the Tivoli remote access account.

Specifies the account name to be used for the Tivoli remote access account. When you use this option, you are prompted for the password.

Specifies an account and password other than root for installing each managed node. When you use this option, you are prompted for the password.

Specifies that the installation should proceed without confirmation. By default, the command identifies the actions that must be taken to perform the installation and requests confirmation before continuing. Using this option, the command identifies the actions and performs the installation without requesting confirmation.

Indicates the key-value pairs that control the installation. These variables can be set or defaulted on the command line. These variables specify required information or override default information.

Note: Type the names of the installation options exactly as specified in the product documentation. Installation options are case sensitive.

Several of the installation variables specify the directories where the managed node will be installed. If a directory contains files from a previous installation, the command does not install these files again. However, you can force any of these directories to be reinstalled by using an exclamation mark (!) character as the value for the variable.

The following are the installation variables related to the installation directories:

**BIN=binaries**
Overrides the default installation path (/usr/local/Tivoli/bin) for the binaries.

**LIB=libraries**
Overrides the default installation path (/usr/local/Tivoli/lib) for the libraries.

**DB=client_database**
Overrides the default installation path (/var/spool/Tivoli) for the object database.

**MAN=manpage**
Overrides the default installation path (/usr/local/Tivoli/man) for the manual pages.

**APPD=X11_defaults**
Overrides the default installation path (/usr/lib/X11/app-defaults) for the X11 application defaults.
CAT=message_catalog
Overrides the default installation path (/usr/local/Tivoli/msg_cat) for the message catalogs.

The following are other useful installation variables:

@AutoStart@=0 | 1
Indicates whether the Tivoli daemon should be started (1) at system boot time. By default, the daemon is not started (0).

@SetPort@=0 | 1
Indicates whether to configure or not configure (0) the remote start capability of the Tivoli daemon. Enabling remote start requires changes to system files; for example, /etc/inetd.conf and /etc/services on Linux and UNIX systems. By default, this capability is configured (1).

@CreatePaths@=0 | 1
Indicates whether to create the specified directories if they do not already exist. By default, directories are created (1). It is an error if a directory you specified with install_variables does not exist.

@ClientAddNoTrans@=yes | no
Indicates whether managed nodes should be added using a transaction. Adding managed nodes without a transaction can save significant time when installing over slow links. If an error occurs, however, you should run the wchkdb command to verify the state of the database.

@ForceBind@=yes | no
Indicates whether communication connections are forced to bind to a single Internet Protocol (IP) address. This option is used in certain high-availability or failover configurations where multiple object dispatchers reside at different IP addresses on a single physical system.

managed_node
Specifies the machines where a managed node is installed. If using the –f option, do not include any machine that is listed in the file specified by this option.

Authorization
install_client or super

Files

Linux and UNIX Files
/tmp/tivoli.cinstall
This file resides on the Tivoli management region server and contains verbose debugging information from all managed node and product installation attempts.

/tmp/install2.cfg.error /tmp/install2.cfg.output
This transient file is created on a managed node during the initialization of its object database. After a successful initialization, these files are removed.
/tmp/client.cfg.error /tmp/client.cfg.output

This transient file is created on a managed node during the configuration of its object database. After a successful initialization, these files are removed.

/etc/Tivoli/setup_env.sh

This file contains useful Bourne shell environment variables. The file can be sourced in from Bourne shell compatible shells after installation.

Note: When you install Tivoli Management Framework on a UNIX operating system, the /tmp/.tivoli directory is created. This directory contains files that are required by the object dispatcher process. You should not delete this directory or any of its contents unless explicitly directed to by your Tivoli support provider. You should also ensure that regularly scheduled disk clean-up jobs (cron or Tivoli jobs) do not remove this directory or its contents.

To use a different directory, you must set an environment variable in both the object dispatcher and the shell. After installing Tivoli Management Framework, perform the following steps to set the necessary environment variables:

1. Create a directory. This directory must have at least public read and write permissions. However, define full permissions and set the sticky bit to ensure that users cannot modify files that they do not own.
2. Set the environment variable in the object dispatcher:
   a. Enter the following command:
      ```bash
      odadmin environ get > envfile
      ```
   b. Add the following line to the envfile file and save it:
      ```bash
      TIVOLI_COMM_DIR=new_directory_name
      ```
   c. Enter the following command:
      ```bash
      odadmin environ set < envfile
      ```
3. Edit the Tivoli-provided setup_env.csh, setup_env.sh, and oserv.rc files in the /etc/Tivoli directory to set the TIVOLI_COMM_DIR variable.
4. For HP-UX and Solaris systems, add the following line to the file that starts the object dispatcher:
   ```bash
   TIVOLI_COMM_DIR=new_directory_name
   ```

   Insert the line near where the other environment variables are set, in a location that runs before the object dispatcher is started. The following list contains the file that needs to be changed on each operating system:
   - For HP-UX operating systems: /sbin/init.d/Tivoli
   - For Solaris operating environments: /etc/rc3.d/S99Tivoli
5. Shut down the object dispatcher by entering the following command:
   ```bash
   odadmin shutdown all
   ```
6. Restart the object dispatcher on the Tivoli server by entering the following command:
   ```bash
   odadmin reexec 1
   ```
7. Restart the object dispatcher on the managed nodes by entering the following command:
Windows Files

%TMPDIR%\tivoli.cinstall
This file resides on the Tivoli server and contains verbose debugging information from all managed node and product installation attempts.

%TMPDIR%\install2.cfg.error %TMPDIR%\install2.cfg.output
These transient files are created on a managed node during the initialization of its object database. After a successful initialization, these files are removed.

%TMPDIR%\client.cfg.error %TMPDIR%\client.cfg.output
These transient files are created on a managed node during the configuration of its object database. After a successful initialization, these files are removed.

Examples

1. The following example installs managed nodes dan and barney in policy region bedrock in the default locations. The user is prompted for the installation password and the root password for these managed nodes. The complete path to the installation image is /cdrom. The example also overrides the default location for the object database directory and installs this database in /var/spool/database instead.
   wclient -dIP -c /cdrom -p bedrock DB=/var/spool/database dan barney

2. The following example installs managed nodes sherman and sulphur in policy region austin:
   wclient -d -c /cdrom -p austin sherman sulphur

See Also

wchkdb winstall wrmnode wserver wstandalone
**wclrblk**

Removes a block of statements from a file. This command should be run from an endpoint.

**Syntax**

```
wclrblk [-r] -s "start_string" -e "end_string" [-o output_file] file_name
```

**Description**

The `wclrblk` command removes a block of statements from a file. This command is intended to remove a block of statements clearly delimited at the beginning and end of the block (such as a block added using the `winsblk` or `wrplblk` command). The caller must insert the delimiting lines along with the actual block of statements.

**Options**

- `-e "end_string"`
  Specifies a string to search for that signifies the end of the block of statements in the file. You must surround the string with double quotation marks.

- `-o output_file`
  Specifies the name of the file that receives the processed file. If this option is not specified, output is written to standard output. You cannot redirect the processed file to the file that you are modifying.

- `-r`
  Removes the delimiter lines in addition to the block of statements.

- `-s "start_string"`
  Specifies a string to search for that signifies the start of the block of statements in the file. You must surround the string with double quotation marks.

- `file_name`
  Specifies the file from which the block should be removed.

**Return Values**

- `0`
  Indicates that the command successfully removed the specified block of statements.

- `nonzero`
  Indicates that the command did not successfully remove the specified block of statements.

**Examples**

The following example removes delimiter lines and statement blocks starting with `[keyboard]` and ending with `type=4` from the `c:\windows\system.ini` file, and writes the `c:\temp\output.fil` file:

```
wclrblk -r -s "[keyboard]" -e "type=4" -o c:\temp\output.fil c:\windows\system.ini
```

**See Also**

`winsblk`, `wrplblk`
**wclrline**

Removes a single line from a file. This command should be run from an endpoint.

**Syntax**

```
wclrline [-f] -s "search_string" [-o output_file] file_name
```

**Description**

The `wclrline` command removes a line from a text file, as specified by the search string. By default, output from this command is written to standard output.

**Options**

- `-f`
  Processes only the first occurrence of the search string. If this option is not specified, all lines containing the search string are removed.

- `-o output_file`
  Specifies the name of the file that receives the processed file. If this option is not specified, output is written to standard output.

- `-s "search_string"`
  Specifies a string to search for in the file. If the search string is contained in a line, the line is removed from the file. You must surround the string with double quotation marks.

- `file_name`
  Specifies the name of the file from which to read.

**Return Values**

- `0`
  Indicates that the command successfully removed the specified line.

- `nonzero`
  Indicates that the command did not successfully remove the specified line.

**Examples**

1. The following example removes the first occurrence of a line starting with `[boot]` from the `c:\windows\system.ini` file and writes the output to the `c:\temp\output.fil` file:

   ```
   wclrline -f -s "[boot]" -o c:\temp\output.fil \c:\windows\system.ini
   ```

2. The following example removes all lines that have `device=` in them from the `c:\windows\myapp.ini` file and writes the output to the `c:\temp\output.fil` file:

   ```
   wclrline -s "device=" -o c:\temp\output.fil c:\windows\myapp.ini
   ```

**See Also**

- `winsline`
- `wrplline`
WCO

Checks out Revision Control System (RCS) revisions.

Syntax

wco [options] file

Description

The wco command retrieves a revision from each RCS file and stores it in the corresponding working file.

Paths matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in the wci command.

Revisions of an RCS file can be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (such as compiling) need not be locked. A revision checked out for editing and later check-in must typically be locked. Checkout with locking fails if the revision to be checked out is currently locked by another user. (A lock may be broken with the wracs command.) Checkout with locking also requires the caller to be on the access list of the RCS file, unless the caller is the owner of the file or the super user, or the access list is empty. Checkout without locking is not subject to access list restrictions and is not affected by the presence of locks.

A revision is selected by options for revision or branch number, check-in date and time, author, or state. When the selection options are applied in combination, the wco command retrieves the latest revision that satisfies all of them. If none of the selection options is specified, the command retrieves the latest revision on the default branch (typically the trunk; see the -b option of the wracs command). A revision or branch number can be attached to any of the options -f, -l, -I, -M, -p, -q, -r, or -u. The options -d (date), -s (state), and -w (author) retrieve from a single branch, the selected branch, which is specified by the -f, ..., -u options, or the default branch.

A wco invocation applied to an RCS file with no revisions creates a zero-length working file. The wco command always performs keyword substitution.

The working file inherits the read and execute permissions from the RCS file. Also, the owner write permission is turned on, unless -k v is set or the file is checked out unlocked and locking is set to strict (see the wracs command).

If a file with the name of the working file exists and has write permission, the wco command aborts the checkout, asking beforehand if possible. If the existing working file is not writable or -f is given, the working file is deleted without asking.

Options

-d date

Retrieves the latest revision on the selected branch whose check-in date/time is less than or equal to date. The date and time may be given in free format. The time zone LT stands for local time; other common time zone names are understood.
Most fields in the date and time may be defaulted. The default time zone is UTC. The other defaults are determined in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current time zone values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date 20, 10:30 defaults to 10:30:00 UTC of the 20th of the current month and year of the UTC time zone. The date/time must be quoted if it contains spaces.

\texttt{\textbf{~j joinlist}}

Generates a new revision that is the join of the revisions on \texttt{joinlist}. This option is obsoleted by the \texttt{wrcsmmerge} command but is retained for backwards compatibility.

\texttt{\textbf{~M [rev]}}

Set the modification time on the new working file to be the date of the retrieved revision. Use this option with care; it can confuse the \texttt{make} command.

\texttt{\textbf{~s state}}

Retrieves the latest revision on the selected branch whose state is set to \texttt{state}.

\texttt{\textbf{~w [login]}}

Retrieves the latest revision on the selected branch that was checked in by the user with login name \texttt{login}. If the option \texttt{login} is omitted, the caller login is assumed.

The \texttt{joinlist} is a comma-separated list of pairs of the form \texttt{rev2:rev3}, where \texttt{rev2} and \texttt{rev3} are symbolic or numeric revision numbers. For the initial pair, \texttt{rev1} denotes the revision selected by the options \texttt{~f}, ..., and \texttt{~w}. For all other pairs, \texttt{rev1} denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, the \texttt{wco} command joins revisions \texttt{rev1} and \texttt{rev3} with respect to \texttt{rev2}. This means that all changes that transform \texttt{rev2} into \texttt{rev1} are applied to a copy of \texttt{rev3}. This is particularly useful if \texttt{rev1} and \texttt{rev3} are the ends of two branches that have \texttt{rev2} as a common ancestor. If \texttt{rev1<rev2<rev3} are on the same branch, joining generates a new revision that is like \texttt{rev3}, but with all changes that lead from \texttt{rev1} to \texttt{rev2} undone. If changes from \texttt{rev2} to \texttt{rev1} overlap with changes from \texttt{rev2} to \texttt{rev3}, the \texttt{wco} command reports overlaps as described in \texttt{merge}.

For the initial pair, \texttt{rev2} can be omitted. The default is the common ancestor. If any of the options indicate branches, the latest revisions on those branches are assumed. The options \texttt{~l} and \texttt{~u} lock or unlock \texttt{rev1}.

\texttt{\textbf{~f [rev]}}

Forces the overwriting of the working file; useful in connection with \texttt{~q}. See also “File Modes.”

\texttt{\textbf{~l [rev]}}

Interactive mode; the user is prompted and questioned even if the standard input is not a terminal.

\texttt{\textbf{~k k}}

Generates only keyword names in keyword strings; omits their values. See “Keyword Substitution.” For example, for the \texttt{Revision} keyword, generates
the string $\text{Revision}$ instead of $\text{Revision}: 5.7$. This option is useful to ignore differences due to keyword substitution when comparing different revisions of a file.

-k o Generates the old keyword string, present in the working file just before it was checked in. For example, for the Revision keyword, generates the string $\text{Revision}: 1.1$ instead of $\text{Revision}: 5.7$ if that is how the string appeared when the file was checked in. This can be useful for binary file formats that cannot tolerate any changes to substrings that happen to take the form of keyword strings.

-k kv Generates keyword strings using the default form, such as $\text{Revision}: 5.7$ for the Revision keyword. The name of the person who locked the file (the locker name) is inserted in the value of the Header, Id, and Locker keyword strings only as a file is being locked, for example, by the wci -l and wco -l. This is the default.

-k kv1 Similar to -k kv, except that a locker name is always inserted if the given revision is currently locked.

-k v Generates only keyword values for keyword strings. For example, for the Revision keyword, generates the string 5.7 instead of $\text{Revision}: 5.7$. This can help generate files in programming languages where it is hard to strip keyword delimiters such as $\text{Revision}: $ from a string. However, further keyword substitution cannot be performed after the keyword names are removed, so this option should be used with care. Because of the possibility of losing keywords, this option cannot be combined with -l, and the owner write permission of the working file is turned off; to edit the file later, check it out again without -k v.

-l [rev] Same as -r, except that it also locks the retrieved revision for the caller.

-p [rev] Prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when wco is part of a pipe.

-q [rev] Quiet mode; diagnostics are not printed.

-r [rev] Retrieves the latest revision whose number is less than or equal to rev. If rev indicates a branch rather than a revision, the latest revision on that branch is retrieved. If rev is omitted, the latest revision on the default branch (see the -b option of the wrcs command) is retrieved. If rev is $\$, wco determines the revision number from keyword values in the working file. Otherwise, a revision is composed of one or more numeric or symbolic fields separated by periods. The numeric equivalent of a symbolic field is specified with the -n option of the commands wci and wrcs.

-u [rev] Same as -r, except that it unlocks the retrieved revision if it was locked by the caller. If rev is omitted, -u retrieves the revision locked by the caller, if there is one; otherwise, it retrieves the latest revision on the default branch.

-V n Emulates RCS version n, where n can be 3, 4, or 5. This may be useful when interchanging RCS files with others who are running older versions of RCS. To see which version of RCS your correspondents are running, have them invoke the wrlog command on an RCS file; if none of the first few lines of output contain the string branch: it is version 3; if year is represented by two digits, it is version 4; otherwise, it is version 5. An RCS
file generated while emulating version 3 loses its default branch. An RCS revision generated while emulating version 4 or earlier has a time stamp that is off by up to 13 hours. A revision extracted while emulating version 4 or earlier contains dates of the form yy/mm/dd instead of yyyy/mm/dd and can also contain different white space in the substitution for $\text{Log}$.


-x suffixes

Uses suffixes to characterize RCS files. See the wci command for details.

Keyword Substitution

Strings of the form $\text{keyword}$ and $\text{keyword}:::.$ embedded in the text are replaced with strings of the form $\text{keyword}:\text{value}$, where keyword and value are the pairs listed below. Keywords can be embedded in literal strings or comments to identify a revision. Initially, the user enters strings of the form $\text{keyword}$. On checkout, the wco command replaces these strings with strings of the form $\text{keyword}:\text{value}$. If a revision containing strings of the latter form is checked back in, the value fields are replaced during the next checkout. Thus, the keyword values are automatically updated on checkout. This automatic substitution can be modified by the –k option.

Keywords and their corresponding values are as follows:

$\text{Author}$

The login name of the user who checked in the revision.

$\text{Date}$

The date and time (UTC) the revision was checked in.

$\text{Header}$

A standard header containing the full path name of the RCS file, the revision number, the date (UTC), the author, the state, and the locker (if locked).

$\text{Id}$

Same as $\text{Header}$, except that the RCS file name is without a path.

$\text{Locker}$

The login name of the user who locked the revision (empty if not locked).

$\text{Log}$

The log message supplied during check-in, preceded by a header containing the RCS file name, the revision number, the author, and the date (UTC). Existing log messages are not replaced. Instead, the new log message is inserted after $\text{Log}:::$. This is useful for accumulating a complete change log in a source file.

$\text{RCSfile}$

The name of the RCS file without a path.

$\text{Revision}$

The revision number assigned to the revision.

$\text{Source}$

The full path name of the RCS file.

$\text{State}$

The state assigned to the revision with the –s option of the wrcs or wci command.

Notes

Links to the RCS and working files are not preserved.
There is no way to selectively suppress the expansion of keywords, except by writing them differently. In nroff and troff, this is done by embedding the null character (\&) into the keyword.

**Diagnostics**

The RCS path name, the working path name, and the revision number retrieved are written to the diagnostic output. The exit status is zero if and only if all operations were successful.

**Files**

The `wco` command accesses files much as the `wci` command does, except that it does not need to read the working file.

**Environment Variables**

- `RCSINIT` Options prepended to the option list, separated by spaces. See the `wci` command for details.

**Defects**

Occasionally, the –d option accepts no date before 1970.

**Author**


**See Also**

**wconnect**

Connects two Tivoli regions.

**Syntax**

```
wconnect [-u] [-P port] [-m mode] [-r encrypt_level] -s server region_num
wconnect [-n] [-u] [-P port] [-c encrypt_level] [-l login] [-m mode] [-r encrypt_level]
```

**Description**

The `wconnect` command establishes a connection between the Tivoli servers in two Tivoli regions. The first usage of `wconnect` shown in the syntax performs a secure connection. To complete the connection, this command must be run on both the Tivoli servers being connected. The second usage performs a remote connection. This command is run on only one Tivoli server. If the connection being made is to be a one-way connection, the Tivoli server from which the remote connection was performed becomes the managing server.

Tivoli regions can be connected in a one-way connection or a two-way connection. With a one-way connection, one server is the managing server and one is the managed server. An administrator on the managing server can manage any resources on either the managing server or the managed server. However, an administrator on the managed server cannot manage resources on the managing server. With a two-way connection, administrators on either server can manage any resources of the other server.

**Options**

- **-c encrypt_level**
  Specifies the interregion encryption level that is in use in the local Tivoli region. Valid encryption levels are `simple`, `none`, or `DES`. If the encryption level is `none`, no key is required. If the level is `simple` or `DES`, the command prompts for the key in use. The encryption key is the same as the installation password specified during the Tivoli server installation. The encryption level must be the same as that specified during the Tivoli server installation. The default encryption level is `simple`.

- **-l login**
  Supplies a login name for the remote connection process. This login must be a valid login for a user on the remote server, and the user must have a Tivoli administrator with the `super` role defined in the remote Tivoli region. If the `-l` option is specified, the command prompts for a password. If the trusted host facility is used, do not type the password. Instead, press the Enter key to continue.

- **-m mode**
  Specifies the mode of connection to be established between Tivoli regions. Valid connection modes are as follows:

  **Two-way**
  Establishes a two-way connection between Tivoli regions. In a two-way connection, both Tivoli servers have managing authority over the resources in both regions. This is the default value.
wconnect

Managing
Establishes a one-way connection with the local Tivoli server as the managing server. The local Tivoli server can manage resources in the remote Tivoli region, but the remote Tivoli server cannot manage resources in the local region.

Managed
Establishes a one-way connection with the local Tivoli server as the managed server. This option is valid on secure connections only. For remote connections, the local server can be only a managing server.

-\n Instructs Tivoli Management Framework not to prompt for passwords. You can use this option only when you have trusted host access and do not require an encryption key.

Note: Because a Windows operating system does not have trusted host access, this option cannot be used when connecting Tivoli servers on Windows operating systems.

-P port
Specifies the port number to use for communication with the Tivoli server if different from the local object dispatcher port.

Note: This option is meant for use in development and test environments only. It should not be used in a production environment.

-r encrypt_level
Specifies the encryption level in use in the remote Tivoli region. Valid encryption levels are simple, none, or DES. If the encryption level is none, no key is required. If the level is simple or DES, the command prompts for the key in use. The encryption key is the same as the installation password specified during the Tivoli server installation. The encryption level must be the same as that specified during the Tivoli server installation. The default encryption level is simple.

-s
Establishes the connection using the secure connection process. This requires running the command with the -s option on each connected Tivoli server.

-u
Updates resources between Tivoli regions in a two-way connection or from the remote region in a one-way connection. Note that because this is a time-intensive process, you can use the wlsconn command with the -u option to perform this operation at a later date (not connection time).

region_num
Specifies the region number of the remote Tivoli region.

server
Specifies the name of the Tivoli server in the remote region.

Authorization
super

Examples
1. The following example performs a remote two-way connection. Both Tivoli servers use simple encryption. The administrator running this command uses sally as a login on the remote server, elbert. Resource information is exchanged between the servers as soon as the connection is complete.
wconnect -u -c simple -l sally -m Two-way -r simple elbert

2. The following example performs a remote one-way connection. The local server uses no encryption, but the remote server, cook, uses simple encryption. The local server becomes the managing server and the remote server becomes the managed server. Resource information is updated on the local server.
   wconnect -u -c none -m Managing -r simple cook

3. The following example performs a secure two-way connection. The remote server, pinatubo, uses simple encryption (by default). The region number of the remote server is 4004418954. Resource information is exchanged between the servers when the connection has been completed.
   wconnect -u -s pinatubo 4004418954

To complete the connection, you must run the wconnect command on the remote server also. The following command must be run from pinatubo. In this example, the encryption level is specified with the -r option, while the previous command used the default value. Resource information is updated between servers.
   wconnect -u -r simple -s everest 4004598145

4. The following example performs a secure one-way connection. The remote server, meiron, uses simple encryption (by default). The region number of the remote server is 0003432265. The local server will be the managed server. Resource information is exchanged between the servers when the connection is complete.
   wconnect -u -m Managed -s meiron 0003432265

To complete the connection, you must run the wconnect command on the remote server also. The following command must be run from meiron. meiron will be the managing server.
   wconnect -u -m Managing -s space 4004598145

Note: If you make a one-way connection using the secure connection process and the first server you run the wconnect command on is specified as the managing server instead of the managed server, you must run the wlsconn command with the -u option to complete the exchange of connection information on the managing server.

See Also

| wdisconn | wlsconn | wupdate |
**wcpcdrom**

Copies installation images from a CD to a system directory.

**Syntax**

```
wcpcdrom [-i start_index] [-a] [-c] [-n] cdrom_list new_cd_dir [interp_list]
```

**Description**

The **wcpcdrom** command copies Tivoli installation images from a Tivoli CD into a system directory, *new_cd_dir*. Using this command, you can copy the contents of a CD to a system directory, merge multiple CD images in a system directory, or copy a single interpreter type from one or more CDs to a system directory. If, for example, you want to build an installation image for the HP-UX version of Tivoli Management Framework and all the Tivoli applications, you could use this command to create a single directory that contains only the HP-UX 10.01 binaries and libraries for all the Tivoli applications you want to install. You can then install all the applications from that directory.

By default, the **wcpcdrom** command creates symbolic links to the images on the CD. If you are copying images from multiple CDs, you should use the `-c` option, which copies the files instead of creating the symbolic link.

When copying images from multiple CDs, you must also use the `-i` option. This option indicates the number that the **wcpcdrom** command should start with when numbering packets in the system directory. For example, suppose you have already copied one Tivoli CD to a directory. The directory contains files file1.pkt through file95.pkt. When you copy the second CD, you must use the `-i` option to avoid overwriting the existing files. You might, for example, specify wcpcdrom -i 96 to start the numbering packets on the second CD sequentially with the file96.pkt file.

When *new_cd_dir* contains the images you want, you can copy the full image onto a tape using the following command:

```
cd new_cd_dir
tar -cvhf tape_device
```

To install from the new image, create an installation directory and run the **wppreinst** command in the same way that you would install from the CD. See *Tivoli Enterprise Installation Guide* for installation procedures.

**Options**

- `-a` Copies all files and directories from the CD.
- `-c` Copies the CD images instead of creating symbolic links to the CD.
- `-i start_index`
  Indicates the number at which the command should begin numbering packets copied into *new_cd_dir*. The *start_index* variable can be any number 1 through 9999.
- `-n` Shows what the command does when invoked. No files are changed when this option is specified.
- `cdrom_list`
  Identifies the path to the CD. Multiple paths must be separated by commas.
**interp_list**

Identifies the interpreter type to be copied from the CD. Multiple interpreter types must be separated by a space. If omitted, all interpreter types are copied. For a complete list of valid interpreter types, see the Tivoli Management Framework Release Notes.

**new_cd_dir**

Identifies the directory into which the file packets are copied.

**Authorization**

No special authorization is required to use this command.

**Examples**

1. The following example copies the contents of the cdrom1 CD to the /tmp/tiv_install directory:
   
   ```
   wcpcdrom -c /mycdrom/cdrom1 /tmp/tiv_install
   ```

2. The following example copies only the solaris2 interpreter type from the cdrom2 CD to the /tmp/tme_install directory with file packets being numbered sequentially starting at 96:
   
   ```
   wcpcdrom -c -i 96 /mycdrom/cdrom2 /tmp/tme_install solaris2
   ```

3. The following example copies the HP-UX versions of Tivoli Management Framework and a Tivoli application from the /mycdrom/tmp_cdrom and /urcdrom/apps_cdrom directories, respectively, to the /home/new_dir directory with file packets being numbered sequentially starting at 1:
   
   ```
   wcpcdrom -c -i 1 /mycdrom/tmp_cdrom, /urcdrom/apps_cdrom \ /home/new_dir hpux
   ```
wcpyfile

Enables a NetWare configuration program (.NCF) to copy a file. This command should be run from an endpoint.

Syntax

```
 wcpyfile -s source_path -d destination_path
```

Description

The `wcpyfile` command was created to support .NCF configuration programs. This command allows you to copy files from one volume or directory to another on the NetWare machine from within an .NCF configuration program.

Options

```
-d destination_path
Specifies the full path to the destination file.

-s source_path
Specifies the full path to the source file.
```

Examples

The following example copies the SYS:TEMP\FILE.NLM file to the SYS:SYSTEM\FILE.NLM file within the .NCF configuration program:

```
 wcpyfile -s SYS:TEMP\FILE.NLM -d SYS:SYSTEM\FILE.NLM
```
**wcrtadmin**

Creates a new Tivoli administrator.

**Syntax**

```
wcrtadmin -l login... [-n noticegroup]... [-r group,role1:role2...] [-u user_id [-g group_id]]... name
```

**Description**

The `wcrtadmin` command creates a new Tivoli administrator.

**Options**

- `-g group_id`
  Sets the principal group ID for the new administrator.

- `-l login`
  Sets up a Tivoli login for the new administrator. The login can be entered in one of the following forms:
  - `user_name`
  - `user_name@ManagedNode`
  - `domain\user_name`
  - `domain\user_name@ManagedNode`

  If only `user_name` or `domain\user_name` is specified, the user can log in from any system in the Tivoli management region. If `user_name@ManagedNode` or `NTdomain\user_name@ManagedNode` is specified, the user can log in only from the specified host.

- `-n noticegroup`
  Sets up a notice group subscription for the new administrator. You can specify more than one notice group.

- `-r group,role1:role2`
  Gives the new administrator roles in the specified group. This option must be entered in the form of `@group,role1:role2` (for example, `@Administrator,admin:user` or `@DefaultRegion,super`). To specify a role in a single instance of a group, specify the group type followed by the instance name. Both names must be colon separated. For example, `@PolicyRegion:MyRegion,super:senior` specifies `super` and `senior` roles in policy region `MyRegion`. To specify Tivoli region roles, use the string "global" (for example, `-r global,user`).

- `-u user_id`
  Sets the principal user ID for the new administrator. `user_id` can be entered in one of the following forms:
  - `user_name`
  - `user_name@ManagedNode`
  - `domain\user_name`
  - `domain\user_name@ManagedNode`

**Note:** All possible values for `ManagedNode` are listed under the `Hostname(s)` column in the output of `odadmin odlist`.

**name**

The label of the new administrator. This label displays with the
Administrator icon in the Administrators window. Administrator names can include any alphanumeric character, an underscore (_), a dash (-), a period (.), and a space.

Authorization

senior

Examples

1. The following example creates an administrator with the Tivoli login kimiko on system ohio, membership in the Tivoli-Authorization notice group, the admin authorization role in the Testing policy region, and the label Kimiko:

   wcrtadmin -l kimiko@ohio -n Tivoli-Authorization
   -r @PolicyRegion:Testing,admin Kimiko

2. The following example creates an administrator under the principals callahan@sthelens and callahan@dogma with the roles user in the Tivoli region, the roles super, admin, and user in DefaultRegion, the roles super and senior in group Administrators, and the roles super, senior, admin, user, and backup in the MyRegion policy region. callahan is added as a member of the Tivoli Authorization group. The administrator runs as principal user ID callahan and principal group ID staff. The administrator label is Steve Callahan.

   wcrtadmin -l callahan@sthelens -l callahan@dogma
   -n "Tivoli Authorization"
   -r global,user -r @DefaultRegion,super:admin:user
   -r /Administrators,super:senior
   -r @PolicyRegion:MyRegion,super:senior:admin:user:backup
   -u callahan -g staff "Steve Callahan"

See Also

wgetadmin, wsetadmin
**wcrtgate**

Creates an endpoint gateway.

**Syntax**

```
```

**Description**

The `wcrtgate` command creates a new endpoint gateway on the specified managed node. You must specify at least one of these options.

When you create a gateway, the gateway HTTPd service is disabled by default. To access gateway information from a browser, you need to use the `wcrtgate` command to define the access account and password and enable the gateway to accept HTTP request.

**Options**

- `-h managed_node`
  Specifies the name of the managed node on which the gateway is created. If this option is not used, the gateway is created on the managed node from which the command was given.

- `-i IPX_socket`
  Specifies the port on which the gateway listens for Sequenced Packet Exchange (SPX) packets. The gateway also listens for Internetwork Packet Exchange (IPX) packets on the specified port minus one.

- `-n gateway_name`
  Specifies the name of the gateway. If this option is not used, the gateway name is in the format `managed_node-gateway`.

- `-p TCPIP_port`
  Specifies the Transmission Control Protocol / Internet Protocol (TCP/IP) port number through which the gateway communicates with its assigned endpoints. The default is port 9494.

  **Note:** Do not use the same port number used by the proxy managed node or by the endpoints.

- `-P protocols_list`
  Specifies the supported protocols for the specified gateway. Multiple protocols must be separated by commas, for example, `protocol,protocol`.

- `-t default_session_timeout`
  Sets the session timeout period used during downcalls from the gateway to an endpoint. The specified time represents the amount of time the gateway waits for a response from the endpoint.

**Authorization**

- `senior`

**Examples**

1. The following example creates a gateway on managed node pearl. The resulting gateway is named pearl-gateway and communicates on default port 9494.
wcrtgate

wcrtgate -h pearl

2. The following example creates the gems gateway on managed node diamond:
   wcrtgate -h diamond -n gems

3. The following example creates the subnet30-gateway gateway on managed node vernon that communicates through port 8432:
   wcrtgate -h vernon -n subnet30-gateway -p 8432

4. The following example creates an endpoint gateway on managed node vernon that supports both TCP/IP and IPX protocols. It communicates with TCP/IP endpoints on port 9999 and with IPX endpoints on port 6000:
   wcrtgate -h vernon -P TCPIP,IPX -p 9999 -i 6000 -n gems

See Also

wdelgate
wcrtjob

Creates a job in a task library.

Syntax

wcrtjob -j job_name -l library_name -t task_name -M mode [-s interval -n number] 
-m timeout -o output_format [-D | -d node_name -f file_name] [-h node...] [-p profile_manager...]

Description

The wcrtjob command creates a job using the specified task. A job is a resource in the Tivoli environment that can be run repeatedly.

Options

-d node_name
Specifies the managed node on which to save to the job output.

-D
Displays the job output to the screen.

-f file_name
Specifies the file name in which to save the job output. Each output file name should be different. If the same name is used for multiple operations and these operations are run around the same time, all output is written to this file.

-h node...
Specifies the nodes (managed node or endpoint) on which to run the job. The –h option must be repeated for each node specified.

Note: When an endpoint and a managed node exist with the same name, the –h option defaults to the managed node. To force the –h option to specify an endpoint, use the following syntax, where ep_name is the name of the endpoint:

-h @Endpoint:ep_name

-j job_name
Specifies the name of the job being created. The first character of the name must be an alphabetic character. The remaining characters can be alphanumerics, underscores (_), hyphens (-), periods (.), or spaces.

-l library_name
Specifies the task library containing the task to be included in the job.

-m timeout
Specifies the amount of time in seconds that the task library waits for results to be returned from the task. This option does not affect the execution of the job. If you are using the –M staged option, the timeout must be smaller than the interval time.

-M mode
Specifies the mode in which the job runs. Valid options are as follows:

parallel
Runs the job on all specified nodes and any subscribers simultaneously.

serial
Runs the job on one node at a time.
staged  Runs the job on groups of nodes at specified intervals. If you
         specify this mode, you must use the –n and –s options.

- n  number
         Specifies the number of nodes (managed nodes and endpoints) to run the
         job in each stage. You must specify a value for this option if you use the
         –M staged option. The maximum number is 150.

-o  output_format
         Defines the format of the job output. The job output contains a summary
         of the job on each managed node. Task execution output format is
         specified with an octal number from 0 to 17. The format is constructed by
         adding the value of the desired output. For example, to print the return
         code and standard error, enter -o 12. Output values are as follows:

         01  Prints a descriptive header for each record.
         02  Prints the return code.
         04  Prints the standard output.
         10  Prints the standard error.

-p  profile_manager...
         Specifies the profile managers on which the job runs. The –p option must
         be repeated for each profile manager specified.

-s  interval
         Specifies the number of seconds between when the task runs on one group
         of nodes and when it runs on the next group. You must specify a value for
         this option if you use the –M staged option.

-t  task_name
         The name of the task to include in the job.

Authorization
         admin, senior, super

Examples

1. The following example creates a job named date_job. The job includes the
   date_task task, which is contained in the my_tasks task library. The job runs in
   parallel on managed nodes that subscribe to the marketing profile manager.
   The job runs on each node for 120 seconds before it times out. The output is
   displayed to the screen.
   wcrtjob -j date_job -t date_task -l my_tasks -M parallel -m 120 -p marketing \ 
           -o 017 -D

2. The following example creates a job named date_job2. The job includes the
   date_task task, which is contained in the my_tasks task library. The job runs in
   parallel on node bald. The job runs for 120 seconds before it times out. The
   output is saved to the /tmp/date_job2.output file on managed node bald.
   wcrtjob -j date_job2 -t date_task -l my_tasks -M parallel -m 120 -h bald \ 
           -o 017 -d bald -f /tmp/date_job2.output

See Also
         wcrttask, wdeljob, wruntask
wcrtpol

Creates a new policy object for a class.

Syntax

wcrtpol [-d] class name [parent]

wcrtpol -v class name [parent]

Description

The wcrtpol command creates a new policy default or policy validation object for a class. The new policy object inherits its first set of methods and attributes from an existing class policy object. Use the wgetpolm and wputpolm commands to customize the new policy object.

Options

-d Creates a policy default object. This action is the default unless the –v option is present.

-v Creates a policy validation object.

class The class for the new policy object.

name The name for the new policy object. This names can include any alphanumeric character, an underscore (_), a dash (-), a period (.), and a space.

parent The label of an existing class policy object from which the new policy object inherits its initial methods and attributes. The default parent is the default policy object for the class.

Authorization

senior, super

Examples

This example creates the Restricted policy validation object for ProfileManager that inherits from the BasicProfileManager policy object.

wcrtpol -v ProfileManager Restricted BasicProfileManager

See Also

wchkpol wcrtpol wdelpol wdelpr wgetdfpol wgetpolm wls wlspolm wputpolm
wcrtpr

Creates a policy region.

Syntax

wcrtpr [–a admin...] [–s region] [–m resource...] name

Description

The wcrtpr command creates a policy region. When this policy region is created, policy validation is disabled for all resource types. To enable policy validation, use the wsetpr command.

Options

–a admin...

Adds the new policy region to the specified administrator desktop. To add to multiple desktops, specify –a admin for each administrator.

–m resource...

Specifies a resource to be added to the new policy region list of managed resources. To add multiple resources, specify –m resource for each resource. Using this option requires the policy authorization role.

–s region

Creates the policy region as a subregion of the specified policy region. The new policy region inherits the supported classes of its parent region. When omitted, the policy region is created as a top-level policy region.

name

The name of the new policy region. Policy region names can include any alphanumeric character, an underscore (_), a dash (-), a period (.), and a space.

Authorization

senior but when using the –m option, senior and policy.

Examples

1. The following example creates a top-level policy region and automatically adds it to the root administrator desktop:

wcrtpr -a Root_ceridwen-region new-region

2. The following example creates a subregion under the default region and makes the ManagedNode resource type a managed resource:

wcrtpr -s /Regions/test-region -m ManagedNode node-region

See Also

wchkpol wcrtpol wdelpol wdelpr wgetdfpol wgetpolm wlspolm wputpolm
wcrtprf

Creates a new profile or clones an existing profile.

Syntax

wcrtprf [-c source] profile_manager type profile_name

Description

The wcrtprf command creates a configuration profile of the specified profile type. The profile is created in the specified profile manager.

When source is specified, the command clones an existing profile of the same type. When omitted, the new profile will be initialized empty.

Options

-c source
Specifies the name of an existing profile from which to clone the new one. Valid formats are as follows:

- @profile_name
- @profile_type:profile_name
- /region/policy_region_name/profile_manager_name/profile_name

profile_manager
Specifies the name of the profile manager in which to create the profile. Valid formats are as follows:

- @profile_manager_name
- @ProfileManager:profile_manager_name
- /region/policy_region_name/profile_manager_name

profile_name
Specifies the name of the new profile. Profile names can include any alphanumeric character, an underscore (_), a dash (-), a period (.), and a space.

type
Specifies the type of profile to create. The resource type of the profile is application-specific. Refer to the application-specific documentation for profile-specific details.

Authorization

senior or super

Examples

1. The following example creates the DiskSpace profile in the Development profile manager.
   
   wcrtprf @ProfileManager:Development SentryProfile DiskSpace

2. The following example clones the DiskSpace profile into the Marketing profile manager.
   
   wcrtprf -c @SentryProfile:DiskSpace \ 
   @ProfileManager:Marketing SentryProfile OurDiskSpace
See Also

wcrtprfmg
wdistrib
wgetprf
wgetsub
wlspol
wpopulate
wsub
wunsub
wvalidatem
wcrtprfmgr

Creates a profile manager.

**Syntax**

wcrtprfmgr policy_region profile_manager

**Description**

The `wcrtprfmgr` command creates a profile manager in the specified policy region.

**Note:** To specify whether a profile manager runs in dataless mode, use the `wsetpm` command.

**Options**

- **profile_manager**
  Specifies the name of the new profile manager. Profile manager names can include any alphanumeric character, an underscore (_), a dash (-), a period (.), and a space.

- **policy_region**
  Specifies the policy region in which to create the profile manager:
  - @policy_region
  - @PolicyRegion:policy_region
  - /region/policy_region

**Authorization**

- senior or super

**Examples**

The following example creates the Development profile manager in the Dallas policy region:

wcrtprfmgr @Dallas Development

**See Also**

 nodeList, wdistrib, wgetprf, wgetsub, wlssub, wpopulate, wsetpm, wsub, wunsub, wvalidate
wcrtqlib

Creates a query library.

Syntax

  wcrtqlib policy_region query_library

Description

The wcrtqlib command creates a new query library in the specified policy region.

Options

  policy_region
     Specifies the name of the policy region in which to create the query library.

  query_library
     Specifies the name of the new query library.

Authorization

  senior or super

Examples

  The following example creates the NewQueries query library in the
  amon-sul-Region policy region.

  wcrtqlib amon-sul-Region NewQueries

See Also

  wcrtquery | wdel | wgetquery | wruninvquery | wrunquery | wsetquery
**wcrtquery**

Creates a query in a query library.

**Syntax**

```
wcrtquery [-d description] [-r repository] [-v view] [-c column]... -i [-x] query_library query_name
```

```
wcrqcry [-d description] [-r repository] [-v view] [-c column]... -s [-x] query_library query_name
```

```
wcrquery [-d description] [-r repository] [-v view] [-c column]... -w where_clause [-x]
query_library query_name
```

**Description**

The `wcrtquery` command creates a new query in a query library. You can specify a description for the query, a repository, a view, the columns to query, and SQL search (or where) clauses. You can specify the optional where clauses through either standard input or on the command line.

The defaults for the configuration repository, view, and columns are supplied by the policy defaults in the policy region that contains the query library.

You can view and edit these attributes with the `wgetquery` and `wsetquery` commands or from the Tivoli desktop.

**Options**

- `-c column`
  Specifies the column or columns from which to retrieve information. To include more than one column, use multiple `-c` options. The columns in the output are ordered according to how you enter them here.

- `-d description`
  Specifies a description of the query.

- `-i`
  Reads the nonstructured where clause from standard input.

- `-r repository`
  Specifies the name of the configuration repository from which to retrieve information.

- `-s`
  Reads the structured where clause from standard input. The clause should be in the following format:
  
  `[AND|OR] [NOT] Column_Name {=|!=|<|<=|>|>=|LIKE|IN} Column_Value`

- `-v view`
  Specifies the name of the view or table from which to retrieve information.

- `-w where_clause`
  Provides a where clause on the command line.

- `-x`
  Specifies that the output of the query does not contain duplicate rows.

*query_library*

  Specifies the name of the query library in which to create the query.

*query_name*

  Specifies the name of the new query.
Authorization
admin, senior, or super

Examples
1. The following example creates the DOS-machines query in the NewQuereis query library. This query uses a structured where clause, read from standard input, to find information about machines running DOS. The query looks in the inventory repository in the MACHINE_TYPE view and returns information from the PROCESSOR_TYPE and OPERATING_SYSTEM columns for every instance of BOOTED_OS_NAME that has a value of DOS.

   wcrtquery -d "Find all DOS machines" -r inventory -v \
   MACHINE_TYPE -c PROCESSOR_TYPE -c OPERATING_SYSTEM -s \
   NewQueries \ 
   DOS-machines << \
   EOF \
   (BOOTED_OS_NAME = 'DOS') \
   EOF

2. The following example creates the same query, except that an unstructured where clause specifies that the results will be sorted by HARDWARE_SYSTEM_ID:

   wcrtquery -d "Find all DOS machines" -r inventory -v \
   MACHINE_TYPE -c PROCESSOR_TYPE -c HARDWARE_SYSTEM_ID \ 
   -w "(BOOTED_OS_NAME = 'DOS') ORDER BY HARDWARE_SYSTEM_ID" \ 
   NewQueries DOS-machines

See Also
wcrtqlib wdel wgetquery wruninvquery wrunquery wsetquery
wcrtrim

Creates an RDBMS Interface Module (RIM) object.

**Syntax**

```bash
wcrtrim [-i] -v vendor { -o host_oid | -h host_name } -d database -u user -H db_home
-s server_id [-I instance_home] [-t instance_name] [-a application_label] [-m
max_connections] rim_name
```

**Description**

The `wcrtrim` command creates a new RIM object on a specific managed node. When you use the `-h` or `-o` option, specify a managed node that is local to the Tivoli region where you enter the `wcrtrim` command and where the RIM object will reside. If you do not specify either the name or the object ID of a managed node, the RIM object is created on the Tivoli server.

You cannot change the vendor for a RIM object after it has been created. If you need to change the vendor, you must delete the RIM object and create a new one.

**Notes:**

1. The `-I instance_home` and `-t instance_name` options are used only by DB2 databases.
2. When you specify a path that contains a space, you must enclose the path name in quotation marks (" "). On Windows systems, you can also specify the MS-DOS path. For example, you can specify the path `c:\Program Files\sqlib` in one of the following ways:
   - "c:\Program Files\sqlib"
   - `c:\progra~2\sqlib`

**Options**

- `-a application_label`  
  Specifies the application label for the RIM object.

  **Note:** This argument is not enforced by RIM. It is provided for application use.

- `-d database`  
  Specifies the name of the database (database ID) or data source to which the RIM object will connect.

  **DB2**  
  The DB2 database alias or the local DB2 database name.

  **Oracle**  
  The name of the Oracle instance. This name is the SID option in the configuration connection file. Corresponds to the value in the ORACLE_SID variable.

  **Sybase**  
  The name of the database that the application will use.

  **Microsoft SQL**  
  The name of the ODBC data source that RIM uses to connect to the Microsoft SQL Server database.

  **Informix**  
  The name of the data source defined in the .odbc.ini file.
–h host_name
Specifies the host name of the managed node where the RIM object will reside. The managed node must be in the local Tivoli region. If you do not use this option or the –o option, the RIM object will not be created.

–H db_home
Specifies the full path to the directory where the RDBMS server or client software is installed on the RIM host.

Note: On Windows systems, use one of the following formats when specifying value for a path that contains spaces:
• "c:\Program Files\sqlib"
• c:\progra~1\sqlib
If you do not enclose the value within double quotation marks (") or include the tilde (~) in the path name, the parsing of the command fails.

DB2 The directory where the DB2 software is installed. Equates to the value in the DB2DIR variable.

Oracle The home directory. Equates to the ORACLE_HOME variable.

Sybase
The directory that contains the interfaces file. Equates to the SYBASE variable.

Microsoft SQL
The directory where Microsoft SQL Server is installed.

Informix
The directory where the Informix CLI client is installed. Equates to the value in the INFORMIXDIR variable.

–i
Reads the database password from standard input. If you specify this option, the password can be any length. If you do not specify this option, you are prompted for a password, which must be eight characters or less.

–I instance_home (DB2 only)
Specifies the value of the INSTHOME variable. This value is the home directory where the database instance was created.

–m max_connections
Specifies the maximum number of connections allowed between the RIM object and the RDBMS. It is recommended that the maximum number of connections be the same of the number of threads used by the application.

The range is 1 to 200 connections. The default is 16 connection.

Note: This argument is not enforced by RIM. It is provided for application use.

–o host_oid
Specifies the object ID of the managed node where the RIM object will reside. The managed node must be in the local Tivoli region. If you do not use this option or the –h option, the RIM object will not be created.

–s server_id
Specifies the server ID for the database. This value enables the RIM host to connect to the RDBMS.
DB2  RIM expects the value to be tcpip. Set the value of the DB2COMM variable to tcpip.

Oracle  The name of the Oracle listener service in the tnsnames.ora file. Equates to the value in the TWO_TASK variable.

Sybase  The name of the Sybase server. Equates to the value in the DSQUERY variable.

Microsoft SQL  The name of the host running Microsoft SQL Server.

Informix  The name of the Informix server. Equates to the value in the INFORMIXSERVER variable.

–t instance_name (DB2 only)  Specifies the value of the DB2INSTANCE variable. If the RIM host is the same machine as the database server, this value must be the name of the DB2 server instance. If the RIM host is on a different machine than the database server, this value must be the name of the DB2 client instance.

–u user  Specifies the name of the database user that the RIM object will use to access the database.

DB2  An RDBMS user that has permission to use the server instance and database. The DB2 user ID for the RIM object must match the client instance name, or the instance_name attribute for the RIM object must be set to the client instance name.

Oracle  The RDBMS user that owns the application data space.

Sybase  The RDBMS user that owns the application data space.

Microsoft SQL  The RDBMS user that owns the application data space.

Informix  Must be informix.

–v vendor  Specifies the vendor of the RDBMS you are using. Valid entries are as follows:
  • DB2
  • Oracle
  • Sybase
  • MS_SQL
  • Informix

rim_name  Specifies the label of the RIM object.

Authorization  senior or super in the Tivoli region
Examples

1. The following example interactively creates a RIM object:

   \texttt{wcrtrim -v Oracle -h amon-sul -d amar -u tivoli \}
   \texttt{-H/tivoli/drm/2/amishra/ORACLE -s invdb.world inventory}

2. The following example creates the same RIM object, but reads the password from a file:

   \texttt{wcrtrim -i -v Oracle -h amon-sul -d amar -u tivoli \}
   \texttt{-H/tivoli/drm/2/amishra/ORACLE -s invdb.world inventory \}
   \texttt{< ./passwd}

3. The following example creates the invdh2 RIM object, sets the application label to invdh, and sets the maximum number of RDBMS connections to 10:

   \texttt{wcrtrim -v Oracle -h amon-sul -d amar -u tivoli \}
   \texttt{-H/tivoli/drm/2/amishra/ORACLE -s invdb.world \}
   \texttt{-a invdh -m 10 invdh2}

See Also

\texttt{wdel \ wgetrim \ wsetrim \ wsetrimpw}
wcrttask

Creates a task in a task library.

Syntax

```
wcrttask -t task_name -l library_name [-g group_name] [-u user_name] -r role [-c comments] -i interp_type node_name file_name...
```

```
wcrttask [-F file_name] -t task_name -l library_name [-u user_name] [-g group_name] -r role
```

Description

The `wcrttask` command creates a task in the specified task library. A task is a method you run on specified managed nodes and their subscribers. Each time you run the task you must specify the run information.

If you specify only the `-t` and `-l` options, the task is created in the specified task library but is not executed. To create the task and run it immediately, you must specify the `-i interp_type node_name file_name` option set. You can specify the set multiple times, once for each platform on which the task runs.

Options

```
-c comments
```

Adds any explanatory comments that help identify the task and its purpose.

```
-F file_name
```

Specifies a file that contains information about an existing task. The specified file must be a tar file that is created by running the `wgettask` command. If this option is used, the specified file is imported and a new task is created using the information in the file. This option is useful when importing a task from one Tivoli environment to another.

```
-g group_name
```

Specifies the name of the group under which the task runs.

```
-i
```

Defines the information required to execute the new task on a managed node. You must supply the following values with the `-i` option:

```
interp_type
```

Specifies the interpreter type of the platform on which the task is to be run.

```
node_name
```

Specifies the managed node containing the executable for the specified platform.

```
file_name
```

Specifies the name of the executable to be run on the specified platform.

```
-l library_name
```

Specifies the task library in which to create the task.

```
-r role
```

Specifies the authorization roles required to run the task. Multiple roles can be specified in a colon-separated list, for example, admin:senior:super.
wcrttask

-t task_name
Specifies the name of the new task. The first character of the name must be an alphabetic character. The remaining characters can be alphanumeric, underscores (_), hyphens (-), periods (.), or spaces.

-u user_name
Specifies the name of the user under which the task runs. If you use an asterisk (*) to set user_name to the current user ID (UID), enclose the asterisk in single quotation marks (for example, -u '*)

Authorization
admin, senior, super

Examples
1. The following example creates the date_task task in the my_tasks task library. Administrators must have the super, senior, or user role to run this task. The task runs on the Solaris Operating Environment. The executables for this task are located in /bin/date directory on managed node bald. A comment is also included.
   wcrttask -t date_task -l my_tasks \n   -r super:senior:user -i solaris2 bald /bin/date \n   -c "This task runs the /bin/date command"

2. The following example creates the find_cores task in the my_tasks task library. This task requires the super role. The task runs on the default platform. The executables are located in /tmp/find_cores.sh directory on managed node bald. The task runs as root.
   wcrttask -t find_cores -l my_tasks -r super \n   -i default bald /tmp/find_cores.sh \n   -c "This task finds core files and runs as root" -u root

See Also
wcrtjob, wdeltask, wgettask, wsettask, wtll
**wcrttlib**

Creates a task library.

**Syntax**

```
wcrttlib library_name policy_region_name
```

**Description**

The `wcrttlib` command creates a task library in the specified policy region.

**Options**

- `library_name`
  - Specifies the name of the task library being created. The first character of the name must be an alphabetic character. The remaining characters can be alphanumeric, underscores (_), hyphens (-), periods (.), or spaces.

- `policy_region_name`
  - Specifies the name of the policy region in which to create the task library.

**Authorization**

admin, senior, super

**Examples**

The following example creates the my_tasks task library in the bald-region policy region:

```
wcrttlib my_tasks bald-region
```

**See Also**

`wcrtjob`, `wcrttask`
wdate

Prints the current date and time of the managed node.

Syntax

\texttt{wdate node\_name}

Description

The \texttt{wdate} command prints the current date and time (Greenwich mean time [GMT]) of the specified managed node. The date is printed in the locale-dependent format.

Options

\texttt{node\_name}

Specifies the name of the managed node whose date to print.

Authorization

\texttt{user, admin, senior, super}

Examples

The following example shows the current date and time of the managed node bald:

\texttt{wdate bald}

See Also

\texttt{wdiskspace \ whostid \ wifconfig \ winstdir \ winterp \ wmannode \ wmemsIZE \ wping \ wtimezone \ wuname \ wxterm}
**wdel**

Deletes objects from the Tivoli database.

**Syntax**

```
wdel [–I] label...
```

**Description**

The `wdel` command deletes one or more objects from the Tivoli object database. This command is intended for low-level administration of the Tivoli object database. The default is for the command to fail if a suboperation fails. Tivoli Management Framework provides the following commands for commonly performed actions:

- `wdelep`
  Deletes an endpoint from the Tivoli database.

- `wdeljob`
  Deletes a job from a task library.

- `wdelpol`
  Deletes a default policy object.

- `wdelpr`
  Deletes a policy region.

- `wdelesched`
  Removes a job from the scheduler.

- `wdeltask`
  Deletes a task from the task library.

- `wrmnode`
  Removes a managed node from the Tivoli database.

**Options**

- `–I` Ignores all failed suboperations, which allows the command to continue processing. This option is useful only when multiple labels are passed. This option allows a deletion to fail for individual objects, but the command continues to the next object to be deleted. Without this option, if a deletion fails for an individual object, the command restores any objects already deleted, and then the command terminates with error.

- `label...` Specifies the label of the object to be deleted. This value can be either an object path or a registered name. An object path can be an absolute path (starting at the "/" collection), a relative path (relative to the current working collection), or a simple name (to be found in the current working collection).

**Authorization**

`admin, senior, super`

**Examples**

The following example deletes the profile manager `pm2` from the sevenup-region policy region using its absolute path:

```
wdel /Administrators/wvilburn/sevenup-region/pm2
```
wdel

See Also

wdelep, wdelgate, wdeljob, wdelpol, wdelpr, wdelshed, wdeltask, wrm, wrmnode
wdelep

Deletes an endpoint.

Syntax

wdelep [-d] [-f] endpoint_name ...

Description

The wdelep command deletes the specified endpoints from the Tivoli database. Using the –d option shuts down the endpoint services before deleting the endpoint.

You cannot delete an endpoint across Tivoli region boundaries if the Tivoli regions have a one-way interconnection. To delete an endpoint from a managed, one-way interconnected region, you must first unsubscribe the endpoint from all profile managers in the managing region using the wunsub command. After unsubscribing the endpoint from all profile managers, use the wdelep command in the local Tivoli region to delete the endpoint.

To determine which profile managers an endpoint is subscribed, use the wlssub command.

Options

–d Deletes the lcf.dat file from the endpoint system and shuts down the endpoint service.

–f Forces the endpoint to be deleted regardless of exceptions.

endpoint_name ...

Specifies the names of the endpoints to be deleted. When specifying multiple endpoints, separate each name with a space.

Authorization

senior or super in the policy region containing the endpoint

Examples

The following example deletes endpoint ruby:

wdelep ruby

See Also

wep, wlssub, wunsub
wdelgate

Deletes an endpoint gateway.

Syntax

```
wdelgate gateway_name
```

Description

The `wdelgate` command deletes the specified gateway. After deleting a gateway, run the `wchknode` command to clean up references to the deleted gateway.

Options

```
gateway_name
```

Specifies the name of the gateway to be deleted.

Authorization

`senior`

Examples

The following example deletes gateway gems:

```
wdelgate gems
```

See Also

`wcrtgate`
wdeljob

Deletes a job from a task library.

Syntax

```
wdeljob job_name task_library_name
```

Description

The wdeljob command deletes a job from the task library.

Options

- `job_name`
  - Specifies the name of the job to be deleted.
- `task_library_name`
  - Specifies the name of the task library where the job resides.

Authorization

admin, super, senior

Examples

The following example deletes the date_job job from the my_tasks task library:

```
wdeljob date_job my_tasks
```

See Also

wcrtjob, wdeltask
wdelpol

Deletes a default policy object.

Syntax
wdelpol –d class name
wdelpol –v class name

Description
The wdelpol command deletes the specified policy default object or policy validation object for the resource with the specified label.

Options
–d Deletes the policy default object for the resource. This option is the default if –v is not specified.
–v Deletes the policy validation object for the resource.
class Specifies the label of the class of managed resource whose policy object is to be deleted.
name Specifies the name of the policy object that is to be deleted.

Authorization
senior or super

Examples
The following example deletes the restricted policy validation object for ProfileManager:
wdelpol -v ProfileManager Restricted

See Also
wchkpol, wcrtpr, wdelpr, wgetdfpol, wgetpolm, wslpol, wslpolm, wputpolm
wdelpr

Deletes a policy region.

**Syntax**

```
wdelpr region
```

**Description**

The `wdelpr` command deletes the specified policy region.

**Options**

- `region` Specifies the policy region to be deleted. The policy region must be empty.

**Authorization**

The senior role in the policy region to be deleted.

**Examples**

Each of the following examples deletes the DefaultRegion policy region:

```
wdelpr /Regions/DefaultRegion
wdelpr @PolicyRegion:DefaultRegion
wdelpr @DefaultRegion
```

**See Also**

`wcrtp`
**wdelsched**

Removes jobs from the scheduler.

**Syntax**

```
wdelsched [-b 'mm/dd/yyyy hh:mm'] [-a 'mm/dd/yyyy hh:mm']

wdelsched -s id ...
```

**Description**

The `wdelsched` command removes jobs from the scheduler. If no options are specified, information on all jobs is removed. The `-a` and `-b` options are used to delimit jobs that fall within certain time ranges. The `-s` option is used to specify jobs by their ID number.

**Options**

- `-a 'mm/dd/yyyy hh:mm'`
  - Specifies jobs that are scheduled after this time.
- `-b 'mm/dd/yyyy hh:mm'`
  - Specifies jobs that are scheduled before this time.
- `-s id`...
  - Specifies the job ID.

**Authorization**

super, senior, admin

**Examples**

1. The following example deletes all jobs scheduled to run before May 6, 1998, at 1:00 a.m. and after May 8, 1998, at 1:00 p.m.:
   ```
   wdelsched -b '05/06/1998 01:00' -a '05/08/1998 13:00'
   ```
2. The following example deletes job IDs 876 and 934:
   ```
   wdelsched -s 876 -s 934
   ```

**See Also**

wedsched, wenblsched, wgetsched, wschedjob, wstartsched
**wdeltask**

Deletes a task from a task library.

**Syntax**

```
wdeltask task_name task_library_name
```

**Description**

The `wdeltask` command deletes a task from a task library.

**Options**

- `task_library_name`
  - Specifies the name of the task library where the task resides.

- `task_name`
  - Specifies the name of the task to be deleted.

**Authorization**

`admin`, `senior`, `super`

**Examples**

The following example deletes the `date_task` task from `my_tasks` task library:

```
wdeltask date_task my_tasks
```

**See Also**

`wcrtask`, `wdeljob`
wdepot

Manages MDist 2 repeater depots, which are repositories that store MDist 2 distributions (either temporarily or permanently).

Syntax

```bash
wdepot repeater_name add "id^version" [source_host:]path_name
wdepot repeater_name delete "id^version" "filter"
wdepot repeater_name describe
wdepot repeater_name image "id^version" [source_host:]path_name image_dir
wdepot repeater_name list ["filter"] [-l]
wdepot repeater_name purge
```

Description

The `wdepot` command manages MDist 2 repeater depots. This command provides the ability to add segments to depots, delete depot entries, display depot configuration information, create distribution images, list existing depots, and purge depot entries.

Options

`add "id^version" [source_host:]path_name`

Adds an entry to the depot with the specified segment. Options are as follows:

- **"id^version"**
  Specifies the ID and the version of the file segment. Separate the ID and version with a caret (^) symbol if it contains spaces. The following is an example:
  "Tivoli^3.7.1"

- **path_name**
  Specifies the complete path and file name of the source file. On a Windows operating system, if the path name contains a drive letter (for example, C:\), the `source_host` option must be specified.

- **source_host**
  Specifies the host name for the source file. If not specified, it defaults to the local host.

`delete "id^version" "filter"`

Deletes entries (specified by `filter`) that are currently locked by distributions in the repeater queue. "id^version" specifies the ID and the version of the file segment. Before deletion, the command prompts you for confirmation. The `filter` option supports wildcard characters, such as the asterisk (*).

`describe`

Displays the configuration settings of the depot, including location, size, temporary storage, permanent storage, total storage, and free space. These options can be changed with the `wmdist` command.
image "id^version" [source_host:]path_name image_dir
Creates a distribution image for use when installing from a file server or CD. Options are as follows:

"id^version"
Spefies the ID and the version of the file segment. Separate the ID and version with a caret (^) symbol if it contains spaces. The following is an example:
"Tivoli^3.7.1"

source_host
Specifies the host name for the source file. If not specified, it defaults to the local host.

path_name
Specifies the complete path and file name of the source file. On Windows operating systems, if the path name contains a drive letter (for example, C:\), the source_host option must be specified.

image_dir
Specifies the complete path of the image directory, where the image will be created.

list ['"filter"'] [-1]
Lists all entries in a depot. Use this option to view how much disk space the segments are occupying or to decide whether any segments are no longer needed. You might also list the contents of a depot to remind yourself what you preloaded in it or to verify that the data is still there. Options are as follows:

"filter" Specifies the entries to be listed. If no filter is specified, lists all entries in the depot. The filter option also supports wildcard characters, such as the asterisk (*).

-1 Lists all information for each entry. If not specified, only lists the ID, version, size, percentage completed, and last modification time.

purge
Deletes all entries in the depot, excluding active distributions. The user is prompted for confirmation before the depot is purged. To delete active distributions in a repeater queue, see the wmdist command.

repeater_name
Specifies the label, object ID, or managed node ID of a repeater.

Authorization
The add, delete, image, and purge options require the admin authorization role.

The describe and list options require any Tivoli authorization role.

Examples
1. The following example adds an entry with ID Tivoli and version 3.7.1 to the depot of repeater banshee. The source host is seesaw, and the path name is /data.

wdepot banshee add "Tivoli^3.7.1" seesaw:/data

2. The following example lists all the contents in the depot of repeater banshee:

wdepot banshee list

3. The following example lists detailed information for the program_a_install depot entry with version ver1 on the banshee repeater:
wdepot

wdepot banshee list "program_a_install^ver1" -l

4. The following example lists all entries and entry versions starting with the character T in the depot of repeater banshee:
   wdepot banshee list "T*"

5. The following example lists detailed information for the depot of repeater banshee:
   wdepot banshee list -1

   Entry #1:
   Id: program_a_install
   Version: ver1
   Bytes received: 6755840 (100%)
   Location: /net/futura/programs/proram_a.tar.gz
   Creation time: 2000/04/04 14:11:31
   Last modification time: 2000/04/04 14:11:31
   Receive time: 2000/04/04 14:11:31
   Last access time: 2000/04/04 14:11:31
   Update time: 2000/04/04 14:11:31
   Access count: 0
   Modification count: 1
   Reference count: 1
   Storage status: Permanent

   Entry #2:
   Id: program_a_data
   Version: ver1
   Bytes received: 13511680 (100%)
   Location: /data/program_a_data.tar.gz
   Creation time: 2000/04/18 14:08:46
   Last modification time: 2000/04/18 14:08:46
   Receive time: 2000/04/18 14:08:46
   Last access time: 2000/04/18 14:08:46
   Update time: 2000/04/18 14:08:46
   Access count: 0
   Modification count: 1
   Reference count: 0
   Storage status: Permanent

Note that if you do not specify the -I option, only the ID, version, size, percentage completed, and last modification time are displayed.

6. The following example deletes entries starting with the character T from the depot of repeater banshee:
   wdepot banshee delete "T*"

7. The following example displays the configuration of the depot of repeater banshee:
   wdepot banshee describe

   Depot Location = /usr/local/Tivoli/rpt_dir/depot/
   Depot Size = 512000 (KB)
   Temporary Storage = 0 (KB)
   Permanent Storage = 0 (KB)
   Total Storage = 0 (KB)
   Free Space = 512000 (KB)

8. The following example creates an image of the antivirus file on the banshee repeater and transfers the image to the C:/data directory on a source host named seesaw:
   wdepot banshee image "Tivoli^3.7.1" seesaw:C:/temp/antivirus C:/data

See Also

wmdist
wdepset

Specifies dependency sets that a method needs to run.

Syntax

\texttt{wdepset \textasciitilde c \textit{dependency\_label}} [\textasciitilde C \textit{class}] \texttt{\textasciitilde a \textit{depset nested\_dependency\_set}} | \texttt{\textasciitilde a \textit{tag path} [+p \textit{prefix\_path}] [+a] [+x]} ...

\texttt{wdepset \textasciitilde e \textit{dependency\_set}} \texttt{| \textasciitilde r \textit{tag path}} | \texttt{\textasciitilde r \textit{depset nested\_dependency\_set}} | \texttt{\textasciitilde a \textit{tag path} [+p \textit{prefix\_path}] [+a] [+x]} ...

\texttt{wdepset \textasciitilde d \textit{dependency\_set}}

\texttt{wdepset \textasciitilde v \textit{dependency\_set}}

\texttt{wdepset \textasciitilde r \textit{dependency\_set}}

Description

The \texttt{wdepset} command specifies dependencies that a method needs to run. The command can be used to create, delete, edit, view or resolve a dependency set.

After you specify the dependencies with the \texttt{wdepset \textasciitilde c} command, use the \texttt{wchdep} command to associate the dependency set with a method header.

When you create a dependency set, you include a tag that specifies the type of dependency (for example, \texttt{bin}, \texttt{lib}, or \texttt{${\textsc{interp}}$}).

The \texttt{wdepset} command is usually placed in the object database after script or in the \texttt{.ist} file of an application.

Options

\texttt{\textasciitilde a} Adds a characteristic to a dependency set.

\texttt{+a} Specifies that only the base name of the path subdirectory is appended to the \textit{prefix\_path}.

\textbf{Note}: This option is used only as a modifier to the \texttt{+p} option.

\texttt{\textasciitilde c} Creates a dependency set.

\texttt{\textasciitilde C} The name of the dependency class object if not the Dependency\_Mgr object.

\texttt{\textasciitilde d} Deletes a dependency set.

\texttt{\textasciitilde e} Edits a dependency set.

\texttt{+p} Indicates that the dependent file is not to be deleted from the endpoint when the endpoint cache becomes full.

\texttt{\textasciitilde r} Resolves a dependency set or removes a characteristic from a dependency set.

\texttt{\textasciitilde v} Views a dependency set.

\texttt{+x} Indicates that the endpoint should try to run the dependency before receiving any remaining dependencies and before running the method.
**wdepset**

*dependency_label*

The label to be assigned to the new dependency object. Do not prefix with @DependencyMgr:.

*dependency_set*

The name of the dependency set. You can specify it as a label in the form @DependencyMgr:label or as the object ID (OID).

*nested_dependency_set*

The name of the nested dependency set. You can specify it as a label in the form @DependencyMgr:label or as the object ID.

*prefix_path*

Indicates the path to the file. This is a relative path. Leading slashes (/) are ignored. The directory to which it is relative on the gateway is determined by tag. The directory to which it is relative on the endpoint is determined by the tag and prefix_path. Slashes (/) in path are converted to back slashes (\) on endpoints that require back slashes.

*path*

Indicates the directory on the endpoint to use as the base directory for downloading the dependency, instead of $LCF_CACHEDIR. These dependencies are called out-of-cache dependencies. The prefix path can be an absolute or relative path. If it is a relative path, it is relative to $LCF_DATDIR.

Slashes (/) in paths are converted to back slashes (\) on endpoints that require back slashes.

If the +a option is not specified, the endpoint location is formed by appending the gateway location, relative to the $BINDIR/../lcf_bundle, to prefix_path.

Even when you set prefix_path to the cache directory, the dependency is still considered out-of-cache and not subject to remove when the cache is full.

*tag*

One of the following defined resolve tags:

**bin**

Identifies the dependency as a binary program that resides in the $BINDIR/../lcf_bundle/bin/$INTERP directory of the gateway. The path to the files is resolved at runtime based on the interpreter type of the endpoint. When you specify a dependency as bin, do not include the extension in the wdepset command. Extensions are automatically added when needed.

**depset**

The dependency is another dependency set. This enables you to nest dependency sets.

**lib**

Identifies the dependency as a shared library that resides in the $BINDIR/../lcf_bundle/lib/$INTERP directory of the gateway. The path to the file and the shared library extension are resolved at runtime based on the interpreter type of the endpoint. When you specify a dependency as lib, do no include the extension in the wdepset command.

If you link with a common library on all platforms, you need to make only one entry in the wdepset command. Appropriate extensions are added when the library is downloaded to the endpoint.

It is sometimes convenient for an application to use shared libraries for some interpreter types and static libraries for others.
The **lib** tag supports this. For interpreter types that use static libraries, do not put the library in the directory on the gateway. When the gateway notices the library is not present, it ignores the dependency.

$INTERP

The file is downloaded only for endpoint designed as this interpreter type. When you specify a dependency as $INTERP, include the file extension in the `wdepset` command.

generic

The dependency file is downloaded for each endpoint, regardless of interpreter type. No path resolution is performed. Therefore, you must provide the path to the dependency.

**Authorization**

admin, senior, super

**Examples**

1. The following example creates the `hello_dependency` dependency set and identifies it as the `hello_library` library. The dependency file is downloaded from `$BINDIR/../lcf_bundle/lib/$INTERP/hello_library$SL` on the gateway to `$LCF_CACHEDIR/lib/$INTERP/hello_library$SL` on the endpoint.

   ```
   wdepset -c hello_dependency -a lib hello_library
   ```

2. The following example creates the `my_label_dependency` dependency set and specifies its out-of-cache location in the `sys:/tivoli` directory.

   ```
   wdepset -c my_label_dependency -a nw4 sentry/nw4/sentry.nlm+p sys:/tivoli
   ```

   The dependency set is placed in the `sys:/tivoli/sentry/nw4` directory.

**See Also**

`wchdep`


# wdisconn

Disconnects two Tivoli regions.

## Syntax

```
wdisconn [-s] region_name

wdisconn [-s] -r region_number
```

## Description

The `wdisconn` command disconnects two Tivoli regions. Disconnecting Tivoli regions after they have exchanged resources is a time-consuming process and should be done with care. Keep in mind that this command removes only Tivoli region resources. To remove any objects in collections and to ensure database consistency, always run the `wchkdb` command with the `-ux` options after disconnecting Tivoli regions.

## Options

- `--r region_number`
  
  Specifies the region number of the remote Tivoli region. This option is required if the region name is not available.

- `--s`
  
  Specifies that the connection to be broken is a one-way connection. Do not attempt to connect the other Tivoli server. You might use this option if you inadvertently specified both ends of a one-way connection as the managing server or if you specified the same region number for both ends of a connection. This option disconnects one end of the connection without impacting the other end. You can then reconnect the Tivoli region using the correct information.

- `region_name`
  
  Specifies the name of the remote Tivoli region. The region name is the same as that of the initial policy region created when the server was installed.

## Authorization

`super`

## Examples

1. The following example disconnects region number 4000447345 from the local Tivoli region:
   ```
wdisconn -r 4000447345
   ```

2. The following example disconnects the Tivoli region writers-Region from the local region. Only the writers-Region Tivoli region is disconnected.
   ```
wdisconn -s writers-Region
   ```

## See Also

- `wchkdb`
- `wconnect`
- `wlsconn`
- `wupdate`
**wdiskspace**

Prints available disk space for a managed node.

**Syntax**

```
wdiskspace node_name directory
```

**Description**

The `wdiskspace` command prints the how many kilobytes of disk space are available in the specified directory (or file system).

**Options**

- **directory**
  
  Specifies the directory to check for available disk space. This directory must be an absolute path.

- **node_name**
  
  Specifies the managed node to check for available disk space.

**Authorization**

user, admin, senior, super

**Examples**

The following example shows the available disk space in the `/tmp` directory on managed node bald:

```
wdiskspace bald /tmp
```

11747

**See Also**

`wdate`, `whostid`, `wifconfig`, `winterp`, `wmannode`, `wmemsize`, `wping`, `wtimezone`, `wuname`, `wxterm`
wdistrib

Distributes one or more profile copies.

**Syntax**

```
wdistrib [-l maintain | over_all | over_opts | over_all_no_merge] [-m] [-r] name
[subscriber...]
```

**Description**

The `wdistrib` command distributes one or more copies of a profile to its subscribers. The command updates subscriber databases and configuration files from the Tivoli database. The `name` option specifies the profile to distribute or the profile manager from which all profiles are distributed. The subscribers to distribute to are specified in `subscriber` option. If no subscribers are specified, the command updates all subscribers.

The `-m` option indicates that profiles are distributed to all levels of subscribers. If omitted, profiles are distributed to only the next level of subscribers.

The `-l` option identifies the distribution level. If omitted, the default is `maintain`.

**Note:** The distribution of profiles is based on the applications for which the profile was designed. The options listed might not be valid or applicable to select application-specific profiles.

**Options**

```
-l maintain | over_all | over_opts | over_all_no_merge
```

Specifies the distribution level. The `maintain` option keeps local modifications. The `over_all` option overwrites local modifications. The `over_opts` option merges and distributes all records. The `over_all_no_merge` option distributes only the specified profile.

```
-m
```

Specifies a multistep distribution.

```
-r
```

Sets the return code to 1 if at least one profile distribution or retrieval to or from a profile manager fails.

```
name
```

The name of the profile to be distributed or the profile manager from which all profiles are distributed. Valid formats are as follows:

- `@profile_type:profile_name`
- `/Regions/policy_region_name/profile_manager_name/profile_name`

```
subscriber...
```

Specifies the names of the Tivoli resources to which to distribute a profile copy. Valid formats are as follows:

- `@ManagedNode:node_name`
- `/Regions/policy_region_name/profile_manager_name`

**Authorization**

- admin, senior, super
Examples

1. The following example distributes profiles contained in the Development profile manager to all subscribers of that profile manager. Any local modifications to this profile are maintained.
   
   ```
   wdistrib /Regions/Development
   ```

2. The following example distributes the Admin profile to subscribers pinatubo, rushmore, and to profile manager Marketing. The profile is not distributed to subscribers of Marketing. All local changes are overwritten by this distribution.

   ```
   wdistrib -l over_all @UserProfile:Admin pinatubo \
   rushmore @ProfileManager:Marketing
   ```

3. The following example distributes the Admin profile to the subscribing profile manager Sales. The profile is further distributed to endpoints or profile managers that subscribe to Sales.

   ```
   wdistrib -m @UserProfile:Admin @ProfileManager:Sales
   ```

See Also

```
wcrtprf  wcrtprfmgr  wgetprf  wgetsub  wlssub  wpopulate  wsub  wunsub  wvalidate
```
wdisttask

Controls the distribution of task binaries for a task library.

Syntax

wdisttask –q library_name
wdisttask –s library_name mode
wdisttask –d library_name task_name

Description

The wdisttask command can query or set the distribution mode of a task library or can force a distribution of the binaries for a task to occur.

Task binaries can remain on the Tivoli server of the local region (ALI mode), be distributed to all file servers in the local region (LOCAL mode), or be distributed to all file servers in each connected Tivoli region (GLOBAL mode). You should use the GLOBAL mode only when a Tivoli-based application requires that the task binaries reside on a local file system. A global distribution is a resource-intensive operation that can temporarily slows down your network.

Options

- d Forces the distribution of task binaries to occur immediately.
- q Determines the distribution mode of a task library.
- s Sets the distribution mode of a task library.

library_name
Specifies a task library.

mode
Specifies the mode to be used to distribute a task library. The mode can be one of the following:

ALI Specifies that the task binaries be stored only on the Tivoli server for the local Tivoli region.

GLOBAL Distributes copies of the task binaries to every file server in each connected Tivoli region. If you perform a global distribution to two or more regions that share the same file server, the distribution fails.

LOCAL Distributes copies of the task binaries to every file server in the local Tivoli region.

task_name
Specifies the task binaries to be distributed.

Examples

1. The following example queries the distribution mode of the task library abc:
   wdisttask –q abc
2. The following example changes the distribution mode of the abc task library to LOCAL:
wdisttask -s abc LOCAL

3. The following example immediately distributes the binaries for the task
   rm_core_files in the abc task library:
   wdisttask -d abc rm_core_files

See Also

wcrttask  wgettask
wdskspc

Verifies the amount of disk space available.

Syntax

```
  wdskspc [-q] [-f output_file] [-s required_size] volume_label
```

Description

The `wdskspc` command returns the amount of available disk space on the specified volume. This command can be run from a Windows and NetWare endpoint.

If you specify the `-s` option, the command sets the return code of the command to zero if the required disk space is available and to nonzero if not available. If omitted, displays the total available disk space.

Options

```
  -f output_file
  Redirects any information or errors to the specified output file.

  -q
  Limits the information returned to the required or available disk space depending on whether the `-s` option is specified.

  -s required_size
  Specifies the amount of disk space required. This option can have any of the following suffixes:
  k  kilobytes
  m  megabytes
  g  gigabytes

  The total amount of available disk space is not displayed with this option.

  volume_label
  Returns the amount of available disk space on the volume or disk for the specified volume or disk.
```

Return Values

This command returns one of the following when used with the `-s` option:

```
  0
  Indicates that the command successfully identified that the specified amount of disk space is available.

  nonzero
  Indicates that the required disk space is not available.
```

Examples

1. The following example checks the C: drive for 10 MB of available disk space:
   ```
   wdskspc -s 10m C:\
   ```
2. The following example checks the total disk space available on the C: drive:
   ```
   wdskspc C:\
   ```
3. The following example checks the SYS volume for 20 MB of available disk space on a NetWare machine:
   ```
   wdskspc -s 20m SYS:
   ```
wtmsg

Customizes the message that is displayed when the Tivoli desktop starts.

Syntax

wtmsg {-g | -s "message"}

Description

The wtmsg command allows you to create a message that is displayed when a user starts the Tivoli desktop. The message is displayed in a dialog, and the user must acknowledge this message.

Options

- \(-g\) Returns the message that is displayed when the Tivoli desktop starts.
- \(-s "message"\) Sets the message that is displayed when the Tivoli desktop starts.

Authorization

senior or super

Examples

1. The following example sets the desktop message to "You are entering a secured zone."
   "You are entering a secured zone"
2. The following example removes the defined desktop message:
   ""
3. The following returns the defined desktop message:
   wtmsg -g
weditini

Modifies the groups, variables, and values in an .INI file. This command should be run from an endpoint.

Syntax

```
weditini [-r] [-g section_name [-n variable_name] [-v value] file_name
```

Description

The weditini command edits the contents of an .INI file. Using this command, you can add a variable and value to a section of the file, remove a variable or section, or replace the value of a specified variable.

Options

```
-g section_name
  Specifies the name of the section in the .INI file to process. If you add a variable to a section that does not exist, the section is created and the variable is added.

-n variable_name
  Specifies the variable name to add, replace, or remove.

-r
  Removes the specified section or variable.

-v value
  Specifies the value to add or replace for the variable specified by the -n option.

file_name
  Specifies the full path of the file to edit.
```

Return Values

This command returns one of the following:

0  Indicates that the command edited the .INI file.
nonzero  Indicates that the command did not edit the .INI file.

Examples

1. The following example adds the DefaultDirectory variable to the UserSettings section in the c:\windows\system.ini file and set its value to c:\work directory:
   ```
   weditini -g UserSettings -n DefaultDirectory -v c:\work \ -
   c:\windows\system.ini
   ```

2. The following example removes the UserSettings group from the c:\windows\system.ini file:
   ```
   weditini -r -g UserSettings c:\windows\system.ini
   ```

See Also

wmrgini
wedsched

Edits a job that currently exists in the scheduler.

Syntax

wedsched [–c 'time_period' | OFF] [–C | daytime | nighttime | weekday | weekend] (from to | OFF) [–d desktop...] | OFF] [–f file | OFF] [–h host] [–g group | OFF] [–m email | OFF] [–l label] [–o] [–R 'time_period' | 'iterations' | OFF] [–r 'time_period' | 'iterations' | OFF] [–t mm/dd/yyyy hh:mm'] id

Description

The wedsched command allows administrators to edit a job that currently exists in the scheduler. Administrators must know the ID of the job to edit. This can be found by using the wgetsched command.

Options

–c

Specifies when a job is canceled. Valid options are as follows:

'time_period'

Specifies when a job is canceled if it did not start as scheduled. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, or day. For example, if you specify '3 hour', the job is canceled 3 hours after its originally scheduled start time if it has not already started.

OFF

Turns off the cancellation feature. The job is not canceled.

–C

Specifies the conditions or restrictions under which the job runs. The from option can be either a starting day or a starting time. The to option can be either an ending day or an ending time. Times must be entered using a 24-hour clock (for example, 9:00 for 9 a.m. or 14:00 for 2 p.m.). Days must be entered as numeric values where Sunday is 0 and Saturday is 6. Valid options are as follows:

daytime from to | OFF

Specifies that the job runs only during the day. Specifying the OFF option removes this restriction.

nighttime from to | OFF

Specifies that the job runs only at night. Specifying the OFF option removes this restriction.

weekday from to | OFF

Specifies that the job runs only during the week. Specifying the OFF option removes this restriction.

weekend from to | OFF

Specifies that the job runs only on weekends. Specifying the OFF option removes this restriction.

–d

Specifies whether a Status window is displayed on a desktop when any action is performed on the job. Valid options are as follows:

desktop...

Specifies which desktop displays the Status window when any action is performed on the job. Multiple desktops can be specified.
OFF Specifies that the Status window is not displayed when any action is performed on the job.

-D Disables the job. The job remains in the scheduler but does not run until it is enabled.

-f Specifies whether the job status is written to file when any action is performed on the job. Valid options are as follows:

file Specifies the file to which the job status is written when any action is performed on the job. If a file is specified, the –h option must be used to specify a host on which the file is to be written.

OFF Specifies that the job status is not written to a file.

-g Specifies whether the job status is sent to a Tivoli notice group when any action is performed on the job. Valid options are the following:

group Specifies the notice group to which the job status is sent when any action is performed on the job. Multiple notice groups can be specified.

OFF Specifies that the job status is not sent to a notice group.

-h host Specifies the host on which the job status file is to be written. Must be used with the –f option.

-l label Specifies the name specific to this instance of the job.

-m Specifies whether the job status is sent to an e-mail address when any action is performed on the job. Valid options are as follows:

email Specifies the e-mail address to which the job status is sent when any action is performed on the job. Multiple e-mail addresses can be specified.

OFF Specifies that the job status is not sent to an e-mail address.

-o Specifies that the time indicated in the –t option is in the past. Overrides the warning message.

-r Specifies the repeat information. If the iterations option is specified, the job repeats for a finite number of times. One of the following options must be specified:

'time_period' Specifies how often a job is repeated. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, day, week, month, or year. For example, if you specify '3 hour', the job is repeated every 3 hours.

'iterations' Specifies how many times a job is repeated. You must specify an amount of time, a unit of time, and a number of times to repeat. The unit of time must be minute, hour, day, week, month, or year. For example, if you specify '3 hour 6', the job is repeated every 3 hours until it has been repeated six times.

OFF Turns off the repeat feature. The job is not repeated.

-R Specifies the retry information. If the iterations option is specified, the job retries for a finite number of times. One of the following options must be specified:
"iterations"

Specifies how many times a job is retried. You must specify an amount of time, a unit of time, and a number of times to retry. The unit of time must be minute, hour, or day. For example, if you specify '3 hour 6', the job is retried every 3 hours until it has been retried six times.

"time_period"

Specifies how often a job is retried. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, or day. For example, if you specify '3 hour', the job retries every 3 hours until it completes successfully.

OFF Turns off the retry feature. The job is not retried if it fails to complete successfully.

-t 'mm/dd/yyyy hh:mm'

Specifies the time the job is scheduled to initially execute. You can enter the date and time in either order. You can also enter only the date or only the time. If you enter the time without a date, the job executes at the specified time on the current date. If you enter the date without a time, the job executes at the current time on the specified date. Times must be entered using a 24-hour clock (for example, 9:00 for 9 a.m. or 14:00 for 2 p.m.).

id Specifies the job ID.

Authorization

super, senior, admin

Examples

1. The following example changes the start time of job 782 to 6:00 p.m., November 30, 1998. (Use the wgetshed command to find the job ID.)
   wedsched -t '18:00 11/30/1998' 782

2. The following example edits job 35. The example turns off the cancellation feature and sets the retry for one day after each failure. The job status is not written to a file but is sent to the Tivoli Diagnostics notice group.
   wedsched -c OFF -R '1 day' -f OFF \-g 'Tivoli Diagnostics' 35

3. The following example edits job 728 to run every Monday through Friday:
   wedsched -r '1 day' -C 'weekday 1 5' 728

4. The following example changes the restrictions on job 28 from weekends (Saturday and Sunday) to weekdays (Monday through Friday).
   wedsched -C 'weekend OFF' -C 'weekdays 1 5' 28

See Also

wdelsched wenblsched wgetshed wschedjob wstartsched


**wenblsched**

Disables and enables scheduled jobs.

**Syntax**

```
wenblsched [-b 'mm/dd/yyyy hh:mm'] [-a 'mm/dd/yyyy hh:mm'] [-d]
wenblsched [-s id [-s id]...] [-d]
wenblsched –e
```

**Description**

The `wenblsched` command allows an administrator to disable or enable scheduled jobs. When a job is disabled it does not run as scheduled. If no options are specified, all jobs are enabled or disabled. The `-a` and `-b` options are used to delimit jobs that fall within certain time ranges. The `-s` option is used to specify jobs by their ID number. The `-d` option is used to disable scheduled jobs. The `-e` option is used to synchronize the encryption level of scheduled jobs.

**Options**

- `-a 'mm/dd/yyyy hh:mm'`
  
  Specifies jobs that are scheduled after this time.
- `-b 'mm/dd/yyyy hh:mm'`
  
  Specifies jobs that are scheduled before this time.
- `-d`
  
  Disable jobs.
- `-e`
  
  Synchronizes the encryption level of scheduled jobs with any changes to the encryption level in the Tivoli region done by the `odadmin set_encrypt_level` command. Run the `wenblsched –e` command immediately after restarting all managed nodes in the Tivoli region.
- `-s id...`
  
  Specifies the job ID. You can specify more than one ID.

**Authorization**

`admin`, `senior`, `super`

**Examples**

1. The following example enables all jobs scheduled to run before May 6, 1998, at 1:00 a.m. and after May 8, 1998, at 1:00 p.m.
   ```
   wenblsched -b '05/06/1998 01:00' -a '05/08/1998 13:00'
   ```

2. The following example disables job IDs 529 and 734:
   ```
   wenblsched -s 529 -s 734 -d
   ```

**See Also**

`wdelsched`, `wedsched`, `wgetsched`, `wschedjob`, `wstartsched`
wep

Performs actions on endpoint information contained in the endpoint list.

Syntax

wep

wep endpoint_label

wep endpoint_label get [suboptions]

wep help [option]

wep ls [-d delimiter] [-g gateway_label] [-i suboptions]

wep endpoint_label migrate [-f] gateway_label

wep migrate_to_pref [-a | -d | -n] [-f] [-g gateway_label]

wep endpoint_label set [suboptions]

wep set gateway [-e endpoint_label | -g gateway_label]

wep set interfaces [-e endpoint_label | -g gateway_label] gateway_hostname+port [gateway_hostname+port]...

wep endpoint_label set_config suboptions

wep endpoint_label set_label [-s] new_label

wep endpoint_label status

wep sync_gateways

wep endpoint_label upgrade [disable | enable]

Advanced Options

wep boot_method add tag prototype_object method_name endpoint_oid ...

wep boot_method list tag endpoint_oid ...

wep boot_method remove tag endpoint_oid ...

wep boot_method test tag endpoint_oid ...

wep del endpoint_oid gateway_label ...

wep view endpoint_oid gateway_label ...

Note: Advanced options are described on “Advanced Options” on page 178.

Description

The wep command performs actions on the endpoint information contained in the endpoint list maintained by the endpoint manager. Using this command, you can list the endpoints in a Tivoli region and their assigned gateway, retrieve and set
endpoin t information, migrate an endpoint from one gateway to another, or update other endpoint data within a Tivoli region.

The `wep` command with the `get` option gets information for the specified endpoint, the `wep` command with the `ls` and `-g` options gets information for the specified gateway and its endpoints, and the `wep` command with the `ls` option gets information for all endpoints on multiple gateways.

**Note:** When specifying the object identifier (OID) of an endpoint, remember to append a plus sign (+) as shown in “Examples” on page 179.

**Options**

*get suboption, ...*

Retrieves the specified information about the endpoint named `endpoint_label` from the endpoint list. You can request information about multiple attributes, but you must separate each suboption with a comma.

**Note:** You can abbreviate a suboption provided that the abbreviation is not ambiguous. Instead of specifying `last_login_time`, you can use the abbreviation `last_l`. For this suboption, you cannot use the abbreviation `last`, for it is ambiguous.

The suboptions are as follows:

- **address**
  The network address.

- **all**
  All attributes for the specified endpoint.

- **crypt_mode** `NONE | DES`
  Whether the endpoint uses 56-bit DES encryption when communicating with its gateway. When set to `NONE`, the communication is not encrypted. When set to `DES`, the communication is encrypted.

- **gateway**
  The assigned gateway.

- **httpd**
  The HTTP password. The HTTP password is used to modify endpoint information through a Web browser.

- **id**
  The unique inventory identifier for the specified endpoint.

- **interp**
  The interpreter type or operating system running on the specified endpoint.

- **label**
  The endpoint label.

- **last_login_time**
  The last login time of an endpoint.

- **last_method_time**
  The time of the last method call on an endpoint.

- **last_migration_time**
  The last time the endpoint migrated.

- **login_mode** (Windows operating systems only)
  Whether the endpoint is enabled or disabled for controlling distributions with the Mobile Computing console, and whether a
user can change this endpoint mode. The endpoint is enabled when in mobile mode, but is not enabled when in desktop mode.

mac_address
The MAC address of the endpoint, if Wake on LAN functionality is enabled for the endpoint. If Wake on LAN is disabled for the endpoint, this suboption returns the value NONE.

netload
The current netload setting.

object
The object ID of the endpoint.

policy
The policy region where the endpoint resides. If the endpoint does not reside in a policy region, the result of this command is OBJECT_NIL.

preferred_gateway
The preferred gateway of the endpoint. If the endpoint does not have a preferred gateway, the result of this command is OBJECT_NIL.

protocol
The network protocol used by the endpoint.

subnet_mask
The subnet mask of the endpoint, if Wake on LAN functionality is enabled for the endpoint. If Wake on LAN is disabled for the endpoint, this suboption returns the value NONE.

upgrade_mode
Whether the endpoint is enabled or disabled for upgrades.

version
The version number of the endpoint.

help [option]
Displays the usage information for the specified command option. Without command options, displays the usage statement.

ls [-d delimiter] [-g gateway_label] [-i suboption, ...]
Lists information for gateways and their associated endpoints. The options are as follows:

-d delimiter
Changes the delimiter used in displaying data with the -i option. The default delimiter is a comma.

Note: The HTTP password field does not contain tabs but can contain commas or other commonly used delimiter.

-g gateway_label
Displays information for the specified gateway and its associated endpoints.

-i suboption, ...
Retrieves additional information from the endpoint list for gateways and associated endpoints. Use the suboptions listed for the get option. You can specify multiple suboptions by separating each with a comma.

migrate [-f] gateway_label
Migrates an endpoint, and optionally forces the migration of an endpoint
from its assigned gateway to the specified gateway. There is no communication with the endpoint. Only the endpoint manager and the gateway data are updated. The endpoint detects that it was migrated the next time that there is communication between the endpoint and a gateway. Specifying the –f option forces the migration of the endpoints even when the specified gateway is down.

**migrate_to_pref** [-a | -d | -n] [-f] [-g gateway_label]
Migrates endpoints to their preferred gateway, if the gateway is available. This migration is similar to using the **migrate** option. There is no communication with the endpoint. Only the endpoint manager and the gateway data are updated.

- **–a** Migrates all endpoints to their preferred gateway.
- **–d** Displays a list of endpoints that are not currently assigned to their preferred gateway. No endpoints are migrated.
- **–f** Forces the migration of the endpoints, even when the preferred gateway is down.
- **–g gateway_label**
  Migrates only the endpoints associated with the specified preferred gateway.
- **–n** Migrates all nonmobile endpoints to their preferred gateway.

**set suboption, ...**
Sets the specified information for the endpoint named **endpoint_label**.

**Note:** You can abbreviate a suboption provided that the abbreviation is not ambiguous. Instead of specifying **last_login_time**, you can use the abbreviation **last_l**. For this suboption, you cannot use the abbreviation **last**, for it is ambiguous.

The suboptions are as follows:

**address endpoint_address**
Sets the IP address of an endpoint.

**crypt_mode** **NONE** | **DES**
Specifies whether the endpoint uses 56-bit DES encryption when communicating with its gateway. When set to **NONE**, the communication is not encrypted. When set to **DES**, the communication is encrypted.

**Note:** You must use the **sync_gateways** option to synchronize changes between the endpoint and gateway.

**httpd** [user:password]
Sets the HTTP password for the endpoint. The HTTP password is used to modify endpoint information through a Web browser. You must use the **sync_gateways** option to synchronize changes between the endpoint, gateway, and endpoint manager after making password changes.

**login_mode** **–m** [mobile | desktop] **–s** [variable | constant] (Windows operating systems only)
Sets the login mode of the endpoint. The **–m** option with the mobile value designates a mobile endpoint. The **–m** option with the desktop value designates a nonmobile endpoint. Only users at mobile endpoints can receive and control distributions with the
Mobile Computing console. The –s option with the variable value designates that the user can change the endpoint mode. The –s option with the constant value designates that the user cannot change the endpoint mode.

Note: You must use the sync_gateways option to synchronize changes between the endpoint and gateway.

**netload value**
Specifies the transfer speed for data sent from a gateway to the specified endpoint in bytes per second. If you set this value to zero, this option is ignored. If you set this value to a positive integer, it overrides the wmdist target_netload value for the gateway, but the wmdist net_load value is still observed. If you set this value to any negative integer, it overrides the wmdist target_netload and net_load values, and the gateway sends data to the endpoint as quickly as possible.

**preferred_gateway {gateway_label | nil}**
Sets the preferred gateway to the specified gateway. Use the value nil to clear a preferred gateway.

**upgrade_mode enable | disable**
Sets whether the endpoint can be upgraded.

**set gateway {–e endpoint_label | –g gateway_label}**
Informs the endpoint of its currently assigned gateway. Use the –e option to update the specified endpoint. Use the –g option to update all endpoints assigned to the specified gateway.

**set interfaces {–e endpoint_label | –g gateway_label} gateway_hostname+port gateway_IP_address+port [gateway_hostname+port | gateway_IP_address+port] ...**
Sets the address and port of one or more gateways to which an endpoint can log in. This set of interfaces becomes the gateway list for an endpoint that the endpoint uses when it becomes isolated. Use the –e option to set the interfaces for the specified endpoint. Use the –g option to set the interfaces for all endpoints assigned to the specified gateway. You can specify more than one gateway address. To specify multiple addresses, separate each with a colon.

**set_config suboption**
Allows a user to set endpoint options from any managed node in the Tivoli region. The endpoint configuration information is stored in the last.cfg file on the endpoint. You can only specify one suboption at a time using this command. Depending on the suboption you specify, you might have to to stop and start the endpoint for the change to take effect. The suboptions are as follows:

**allow_proxy_upcalls=true | false**
Specifies whether all communication between the endpoint and its assigned gateway use the configured target port on the endpoint. If set to true, the endpoint and gateway communicate using the target port only. Before endpoint can use proxy upcalls, the endpoint must establish a connection to a proxy-capable gateway. If set to false, the endpoint and its assigned gateway communicate as usual. This option modifies the value of allow_proxy_upcalls in the last.cfg file. The default is false.

**depot_dir=path**
Specifies the directory on the endpoint where multicast
distributions are stored until the distributions are installed. This option modifies the value of depot_dir in the last.cfg file. The default directory is $LCF_DATDIR/depot. If a relative path is specified, the depot directory is relative to the value of the $LCF_DATDIR variable.

detect_address_change=TRUE | FALSE
Specifies whether the endpoint detects changes to its network interface configuration and, if necessary, takes corrective action. When this option is set to TRUE, the endpoint monitors its network interface configuration for address changes. If the listening address for the endpoint changes, the endpoint attempts to log back in to its gateway. (The listening address is set using the local_ip_interface option.) This option applies to Windows 2000, Windows XP, and Windows Server 2003 systems only. The default is FALSE.

diag_interval=seconds
Specifies the interval, in seconds, that an endpoint waits between running a series of self-diagnostic tests. These tests help you determine the ability of an endpoint to run tasks and methods. The tests include checking temporary file space and permissions and cache space and permissions. On Windows operating systems, the tests also check token creation and process spawning. After the tests complete, the results are stored in the lcfd.log file. The results are sent to the current gateway if the endpoint is logged in. If this value is set to zero, a test will only be performed when requested by a gateway. The default is zero. The acceptable range of values is 900 seconds (15 minutes) to 86400 seconds (1 day).

diag_temp_space=bytes
Sets the minimum desired temporary space, in bytes, for running the self-diagnostic tests controlled by the diag_interval option. If this value is set to zero, the tests are not run. The default is zero. The minimum value is 1024.

filefree_upcalls=TRUE | FALSE
Specifies whether consolidated upcalls write the upcall data to disk. This setting is valid only when allow_proxy_upcalls is set to TRUE. In this case, the default is to write the upcall data to disk. However, if you do not want upcall data written to disk, you can set filefree_upcalls to TRUE and all upcall data is transferred through the wire. This option modifies the value of filefree_upcalls in the last.cfg file. The default is FALSE.

lcfd_autostart=TRUE | FALSE
Specifies whether Windows endpoints set the endpoint service to start automatically. When this value is set to TRUE, a Windows endpoint service checks the Windows endpoint service once during endpoint startup and once during endpoint shutdown. If the endpoint service is set to Manual, the endpoint resets the service to Automatic. When this value is set to FALSE, the endpoint does not check the setting of the Windows endpoint service. The default value is FALSE.

lcs.machine_unique_id=ID_string
Identifies the unique identifier of the endpoint. The ID_string value
must be a unique string within the Tivoli environment. The length of the string must be 36 or fewer characters. This value is stored in the $LCF_DATDIR/lcf.id file.

**log_threshold**=integer

Specifies the level of detail written to trace files for the identified endpoint. This option modifies the value of log_threshold in the last.cfg file. The integer value specified can be in the range of 1 to 4, where 1 provides the least level of detail and 4 provides the most. The default is 1.

**login_attempts**=value

The number of directed login attempts on a gateway before the endpoint moves to the next gateway in the list. A directed login attempt is an attempt to log in to either the last known gateway or to a gateway in the login interfaces list. The default is 3.

**login_timeout**=seconds

Specifies the number of seconds that an endpoint waits for a response to a directed login attempt. A directed login attempt is an attempt to log in to either the last known gateway or to a gateway in the login interfaces list. The default value is 300 seconds (5 minutes).

**repair_accts**=TRUE | FALSE

Specifies whether the endpoint tests the installation and configuration of the tmrsrvd account and the Tivoli_Admin_Privileges group and makes repairs as required. When this option is set to TRUE, the endpoint performs this check once during startup, and any corrective actions are noted in the lcfd.log file. When this option is set to FALSE, this check is not performed. The default value is FALSE.

**repair_tap**=TRUE | FALSE

Specifies whether the endpoint tests the installation and configuration of Tivoli Authentication Package and makes repairs as required. When this option is set to TRUE, the endpoint performs this check once during startup, and any corrective actions are noted in the lcfd.log file. When this option is set to FALSE, this check is not performed. The default value is FALSE.

**web_post_interval**=seconds

Specifies the interval, in seconds, at which an endpoint posts event and state change information to a Web server. Information is sent only if an event or state change occurs within this interval. The minimum value is 300 seconds (five minutes). The maximum value is 86,400 seconds (one day). The default value is zero. When this option is set to zero, endpoint Web posting is disabled.

**web_post_url**=URL

Specifies the URL to which an endpoint posts event and state change information. Enter the syntax for URL as shown in the following example:

http://web_server_name:port/relative_path_to_cgi_script/script_name

**Note:** You do not need to specify the port number if you are using the default port of 80.
set_label [-s] new_label
Changes the label for the specified endpoint. Use the -s option to synchronize the gateway and update the endpoint with the new label. This command contacts the endpoint.

status new_label
Lists the status of the specified endpoint. The status can be either "alive" or "endpoint may be unreachable". This command contacts the endpoint.

sync_gateways
Synchronizes the endpoint data stored by the endpoint manager, gateways, and endpoints within a Tivoli region. If you make changes to an HTTP password for the endpoint or policy region, change encryption settings, or decide to use one of the options listed in ["Advanced Options,"] you must use this option for updates to take effect. This option is useful in a scheduled job that updates several endpoint data updates at once.

Note: To change the policy region of an endpoint, use the `wmv` command.

upgrade disable | enable
Specifies whether the endpoint can be upgraded.

display label
Identifies the endpoint where the suboptions are run. Without options, displays all the information for the specified endpoint in a tabular format.

**Advanced Options**
If you are not explicitly familiar with the advanced implications of the `wep` command, call your Support provider before attempting to perform any of the following operations:

```bash
del endpoint_oid gateway_label
```
Deletes the endpoint from an epmgr.bdb file. Because there are separate .bdb files for each gateway, it is necessary to specify the label of the assigned gateway of the endpoint. The gateway label can be found by issuing the `wep` command with the `ls` option. Issuing the `wep` command with the `del` option does not completely delete all references to the endpoint. To delete all references to an endpoint, use the `wdelep` command.

```bash
view endpoint_oid gateway_label
```
Displays endpoint information stored in the .bdb file associated with the gateway and located in the $DBDIR/epmgr.bdb directory. This command does not search the internal cache of the endpoint manager. The gateway label can be found by issuing the `wep` command with the `ls` option.

The following commands are used to configure boot methods created by application developers (using Tivoli Application Development Environment [ADE]) to run on an endpoint. Use these options to configure boot methods during the installation of an endpoint or during the distribution of application profiles. These methods run each time that an endpoint logs in to a gateway.

```bash
boot_method add tag prototype_object method_name endpoint_oid...
```
Adds the specified boot method to the endpoint, where:

```bash
tag
```
Specifies the user specified name.

```bash
prototype_object
```
Specifies the prototype OID, which implements the method.
method_name
Specifies the name of the method.

endpoint_oid
Specifies the object identifiers (OIDs) of endpoints where the method will run. Separate OIDs with spaces.

boot_method list tag endpoint_oid
Lists the prototype object number and method name of the specified boot method on the endpoint.

boot_method remove tag endpoint_oid...
Removes the specified boot method from the endpoint.

boot_method test tag endpoint_oid
Starts the specified boot method on the endpoint.

Authorization

- To view endpoint information with the get, status, and ls options: user or admin
- To set the gateway for an endpoint with the set gateway –e option: senior or super
- To display help information with the help option: user, admin, senior, or super
- To set the HTTP password with the set httpd option: senior or super in the policy region of the endpoint.
- To view the HTTP password with the get httpd option: admin in the policy region of the endpoint.
- To migrate an endpoint, set the gateway for all endpoints, or synchronize endpoint data with the migrate, set interfaces, set gateway –g, or sync_gateways option: senior or super in the Tivoli region

Examples

1. The following example lists the endpoints assigned to the gateway jadams-gateway:
   
   ```
   wep ls -g jadams-gateway
   1849216842.3.522+#TMF_Endpoint::Endpoint# zoomzoom
   1849216842.4.522+#TMF_Endpoint::Endpoint# lilliana-312
   1849216842.5.522+#TMF_Endpoint::Endpoint# miura
   ```

2. The following example returns all information contained in the endpoint list for endpoint ep4624:
   
   ```
   wep ep4624
   object 1769863560.6.522+#TMF_Endpoint::Endpoint#
   label ep4624
   version 41100
   id 2E3B4610-1233-11D8-BE66-9401145B01BC
gateway 1769863560.1.578#TMF_Gateway::Gateway#
pref_gateway 1769863560.1.578#TMF_Gateway::Gateway#
netload OBJECT_NIL
interp w32-ix86
login_mode desktop, constant
protocol TCPIP
address 9.41.21.50+4624
mac address (WOL) 00:03:14:24:c2:09
subnet mask (WOL) 255.255.255.0
policy OBJECT_NIL
httpd tivoli:;jMYe9^x
alias OBJECT_NIL
crypt_mode NONE
   ```
3. The following example returns the network address of endpoint cookG:
   
   ```
   wep cookG get lab,address
   ```
   
   ```
   cookG,146.84.26.26+9494
   ```

4. The following example returns the randomly generated HTTP password for endpoint cookG:
   
   ```
   wep cookG get httpd
   ```
   
   ```
   tivoli:wBHtK'y3
   ```
   where:

   - `tivoli` Represents the user name
   - `:` Is the separator
   - `wBHtK'y3` Represents the password

5. The following example returns the prototype object and method name of boot method test18 on endpoint 1802218143.13.500:
   
   ```
   wep boot_method list test18 1802218143.13.500+
   ```
   
   ```
   Boot Method(s) for Endpoint 1802218143.13.500+
   TagPrototype ObjectMethod Name
   test181802218143.13.500admin
   ```

6. The following example gets the current login mode on endpoint msistrunk:
   
   ```
   wep msistrunk get login_mode
   ```
   
   ```
   desktop, constant
   ```

7. The following example clears the preferred gateway for the endpoint msistrunk:
   
   ```
   wep msistrunk set preferred_gateway nil
   ```

See Also

- `wdelep`
- `wmdist`
- `wmv`
wepmgr

Provides control and configuration for the endpoint manager.

Syntax

wepmgr fsck
wepmgr get
wepmgr help [attribute]
wepmgr ping
wepmgr restart
wepmgr set attribute...
wepmgr start
wepmgr stop
wepmgr test_label endpoint_label
wepmgr update

Description

The wepmgr command provides control and configuration for the endpoint manager. With this command, you can start, stop, and restart the endpoint manager. In addition, this command gets and sets endpoint manager attributes in the Tivoli object database to control endpoint login.

Options

fsck Rewrites the data in the Tivoli name registry endpoint resource from the endpoint data in the endpoint manager.
get Returns a list of endpoint manager object attributes.
help [attribute] Displays the usage statement for the command or information about the specified attribute.
ping Verifies that the endpoint manager is running.
restart Restarts the endpoint manager.
set attribute... Sets the endpoint manager object attributes in the Tivoli object database. Use the get option to return a list of these attributes. Use the update option for attributes to take effect immediately. Attribute values are as follows:

automigrate {off | on | nonmobile} Indicates whether endpoints should migrate to their preferred gateway when the gateway becomes available. This value defaults to off, which indicates that endpoints are not migrated to the preferred gateway. To cause endpoints to migrate, set automigrate...
to **on** or **nonmobile**. When a gateway becomes available, any endpoints that have that gateway as their preferred gateway migrate to the gateway.

**Note:** Mobile endpoints are excluded from this migration if the attribute is set to **nonmobile**. For endpoints that should be excluded from automatic migration, use the `wep` command with the `set preferred_gateway` option to clear the preferred gateway.

**chk_cntl_chars** `value`
When set to 1, disallows control characters in endpoint labels. Invalid endpoint labels are detected in the `allow_install_policy` script. You can overwrite the existing label in this script. The default value is 0. By default, endpoint labels are not checked for control characters.

**ep_hostname_resolution** `0 | 1`
Specifies whether the endpoint manager resolves endpoint hostnames to IP addresses. Endpoint addresses are normally stored as IP addresses or IPX addresses. However, when NAT is enabled (using `odadmin set allow_NAT TRUE`), endpoint addresses can be stored as fully qualified hostnames. By default, the endpoint manager resolves the hostname to an IP address before forwarding the address to be used in endpoint policy scripts and log messages. Setting this attribute to 0 disables hostname resolution. Use this value when endpoint address resolution causes performance problems, for example, when endpoint addresses are not resolvable. Setting this attribute to 1 enables hostname resolution. This is the default value.

**Note:** When this attribute is set to 0, the `endpoint_address` value in the endpoint policy method can be either a hostname or an IP address. For more information about endpoint policy methods, see "Endpoint policy methods" on page 397.

**epmgr_flags** `value`
When set to 1, recaptures orphaned endpoint information. Orphaned endpoints are endpoints that were once a part of the Tivoli management region, but are no longer displayed in the output of the `wep` with the `ls` option. This can occur from a database restore, an accidental deletion, or database corruption. The endpoint is recaptured by the endpoint manager during an isolation login attempt and a new dispatcher number is created for the endpoint. The default is 0.

**invalid_chars** `'regular_expression'`
Specifies characters to be disallowed in an endpoint label during the initial login. The default value is an empty list. Invalid endpoint labels are detected in the `allow_install_policy` script. You can overwrite the existing label in this script.

**labelspace** `'regular_expression'`
Defines the endpoint label name space and forces endpoint labels to conform to a regular expression during initial login. Use the regular expression to specify a set of allowable characters for
endpoint labels (for example, `^[A-Za-z0-9$^_]*$`). If a label does not conform to the specified expression, that endpoint cannot connect to the Tivoli region.

**Note:** Invalid endpoint labels are detected by the **allow_install_policy** script. You can overwrite existing labels in this script.

- **log** _dir_ **directory**
  Specifies the directory on the local system for the epmgrlog and all epmgrlog,timestamp log files.

- **log** _size_ **size** _in_bytes_  
  Specifies the reference size of the epmgrlog file, in bytes. When the referenced size is reached, the epmgrlog file is copied to a epmgrlog,timestamp file and a new epmgrlog file is created. The maximum file size is 10485760 bytes (10 megabytes), and the default file size is 1048576 bytes (1 megabyte).

- **logging** _level_ **value**
  Specifies the level of logging that is written to the epmgrlog file. Each logging level includes the logging information from the lower logging levels. For example, logging level 3 includes the logging information from logging levels zero, 1, and 2. You can specify the following levels of logging using this option:

  0  Only errors and conditions that are probably errors.
  1  Exceptional events that might be errors or warnings.
  2  Other exceptional conditions.
  3  Endpoint manager boot information.
  4  Information about endpoint manager requests and endpoint login attempts. This is the default level.
  5 Verbose login information.
  6 Verbose information about certain endpoint manager requests.
  7  Debug information.
  8 Verbose debug information.

- **login** _interval_ **value**
  This interval is used by gateways and endpoint managers and specifies a time value in seconds in which logins from the same network address are ignored. This interval prevents the effects of multiple initial logins from the same endpoint. The default value is 270 seconds. You must restart the gateways for a new setting to take effect.

In some cases, multiple logins are allowed within one **login_interval**. For example, the logins after the initial login request of an endpoint and after an upgrade are not ignored. For more information about endpoint logins, see *Tivoli Management Framework Planning for Deployment Guide*. To turn off this login filter, set **login_interval** to 0.

- **login** _limit_ **value**
  Specifies the maximum number of concurrent endpoint manager
logins permitted. Endpoint manager logins include isolated, migration, initial, and orphan logins. The default value is 80% of the value of the max_epmgr_rpc_threads option, because approximately 20% of the endpoint manager threads are reserved for methods that are not for logins. Setting this value to zero disallows all logins.

```
max_epmgr_rpc_threads value
```

Specifies the maximum number of threads that the endpoint manager can start. In general, each method that the endpoint manager services requires one thread. Set this option to a value less than the corresponding value for the object dispatcher. (You can view the object dispatcher value using the odadmin get_rpc_max_threads command.) In this way, the endpoint manager will not use up all of the threads of the object dispatcher. The default value for this option is 250.

```
max_after value
```

Specifies the number of after_install_policy scripts that can be run concurrently. The default is 10.

```
max_iom_records value
```

Specifies the number of endpoint entries in each batch that repopulates the name registry. This value is used by the fsck option. The default is 500.

```
max_install value
```

Specifies the number of allow_install_policy scripts that can be run concurrently. The default is 10.

```
max_sgp value
```

Specifies the number of select_gateway_policy scripts that can be run concurrently. The default is 10.

```
migrate_max value
```

Specifies the maximum number of endpoints that can be automatically migrated simultaneously. The default is zero. A value of zero means there is no limit.

```
stanza_interval value
```

Specifies the number of minutes between dumps of information about endpoint login attempts to the epmgrlog file. The default is 720.

```
timeout_interval value
```

Specifies the number of seconds that the endpoint manager waits for a response on certain method calls (for example, calls made to gateways) before the method call times out. The minimum value is 15 seconds. The default value is zero. A value of zero disables this option.

```
start
```

Starts the endpoint manager.

```
stop
```

Stops the endpoint manager.

```
test_label endpoint_label
```

Tests if a label conforms to the currently set labelspace.

```
update
```

Refreshes the attributes in the running endpoint manager from the attributes on the endpoint manager object in the Tivoli object database.
Authorization
You need **senior**, **super** in the Tivoli region.

Examples
1. The following example pings the endpoint manager to verify that it is up and running:
   
   ```
   wepmgr ping
   ```
   
   ep_mgr is running.
2. The following example returns a list of endpoint manager object attributes:
   
   ```
   wepmgr get
   ```
   
   ```
   logging_level = 4
   max_install = 10
   max_sgp = 10
   max_after = 10
   login_interval = 0
   login_limit = 200
   stanza_interval = 720
   timeout_interval = 0
   max_iom_records = 500
   epmgr_flags = 0
   max_epmgr_rpc_threads = 250
   automigrate = off
   migrate_max = 0
   chk_cntl_chars = 0
   labelspace = ''
   invalid_chars = ''
   ```
3. The following example sets the endpoint manager **login_interval** attribute to 0 seconds on the endpoint manager object in the Tivoli object database. By default, endpoints send login requests at 270-second intervals. If you reduce the number of seconds between endpoint initial login request attempts (using the **login_timeout** or **udp_interval** keyword of the lcfd command), you also should reduce the endpoint manager attribute so that endpoint login attempts are not filtered by the gateways and endpoint managers. The following command updates the endpoint manager so that the attribute setting takes effect on the running endpoint manager. Because this value is used by gateways, you also need to restart gateways after issuing this command.
   
   ```
   wepmgr set login_interval 0
   ```
   
   ```
   wepmgr update
   ```

See Also
[wep](#)
**wepstatus**

Returns the status of endpoints.

**Syntax**

```
wepstatus [-o file] [-f format] [-v] [endpoint_name | -a | -g gateway_name | -i file [endpoint_name...]]
```

**Description**

The `wepstatus` command is run from a gateway and returns the status of endpoints. You can specify one or more endpoints for which you want status information, or you can list endpoints in an input file. This command can also return the status for all endpoints associated with a specified gateway or all endpoints in the local Tivoli region.

The endpoint status can be one of the following values:

**connected**

The endpoint is connected to a gateway. The endpoint and gateway can communicate. The endpoint responds to downcalls from the gateway, can run a task, and can initiate an upcall. The gateway services the upcall initiated by the endpoint.

**disconnected**

The gateway knows that the endpoint has disconnected from the gateway and does not try to communicate. The endpoint is logged out of the gateway, but might still be connected to the network.

**unavailable**

The gateway cannot reach the endpoint for one of the following reasons:
- The gateway cannot make a successful downcall.
- The endpoint cannot run a task.
- The endpoint is not servicing upcalls.
- The endpoint is out of disk space.
- There is a permission problem on the endpoint that is preventing the endpoint from reading or writing to a file or running a program.

**unreachable**

The gateway cannot reach the endpoint. The endpoint process might have been stopped or the endpoint might be disconnected from the network.

**unknown**

The gateway associated with the endpoint is down or the gateway has not checked the status of the endpoint. The wepstatus command is unable to determine the status of the endpoint.

The `wepstatus` command can also return the following error codes regarding endpoint status:

- 0 File permission error. Cannot create a temporary file.
- 1 Insufficient disk space in the temporary directory.
- 2 File permission error. Cannot create or update the method cache.
- 3 Insufficient disk space in LCF_CACHEDIR.
- 4 Cannot generate a token for the tmersrvd account.
Cannot generate token for the built-in administrator account.

Cannot spawn a process.

Note: If an endpoint has a status of unavailable and no error code is provided, the endpoint login failed.

Options

- `a` Returns the status for all endpoints in the local Tivoli region.
- `f format` Returns output in the specified format. The valid formats are htm (for HTML format), or csv (for a comma-separated list).
- `g gateway_name` Returns the status for all endpoints associated with the specified gateway.
- `i file` Returns the status for all endpoints contained in the specified file. You must include a fully qualified path and file name. Each endpoint in the file must be listed on a separate line.
- `o file` Writes the status to the specified file. You must include a fully qualified path and file name.
- `v` Specifies verbose output.

`endpoint_name...` Specifies the names of the endpoints for which status is to be returned. Separate each endpoint name with a space.

Authorization

admin, senior, super.

Examples

1. The following example returns the status for the endpoint quartz:

```
weptstatus quartz
```

The output for this command lists the information shown in the following example:

```
Endpoint Label : quartz
Dispatcher Number : 24
Gateway Label : crystal-gw
Interp Type : solaris2
Version : 41100
IP Address : 9.48.125.57+1929
Current Status : unreachable
Status error code(s): -
Last Login Time : 2003/09/24-07:37:41 [Passed]
Last Logout Time : 2003/09/24-07:37:18 [Passed]
Last Downcall Time : 2003/09/24-07:37:42 [Failed]
Last Upcall Time : 2003/09/26-12:07:39 [Passed]
Last EP check Time : 2003/10/03-07:16:03 [Failed]
```

2. The following example returns the status for all endpoints associated with gateway ntgw:

```
weptstatus -g ntgw
```
### wepstatus

The output for this command lists the endpoint label, gateway, interpreter type, endpoint version, object dispatcher number, endpoint status, and error codes, as shown in the following example:

```
Last known status of the endpoint(s):

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Gateway</th>
<th>Interp</th>
<th>Version OD</th>
<th>Status</th>
<th>Error Code(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gem_808_nt02</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>10</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_nt03</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>11</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_nt04</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>12</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_nt05</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>13</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_sol06</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>14</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_sol07</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>15</td>
<td>unknown</td>
</tr>
<tr>
<td>gem_808_sol08</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>16</td>
<td>unreachable</td>
</tr>
<tr>
<td>gem_808_sol09</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>17</td>
<td>unreachable</td>
</tr>
<tr>
<td>gem_808_sol10</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>18</td>
<td>unreachable</td>
</tr>
<tr>
<td>gem_808_sol11</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>19</td>
<td>unreachable</td>
</tr>
<tr>
<td>gem_808_sol12</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>20</td>
<td>unreachable</td>
</tr>
</tbody>
</table>
```

3. The following example returns the status for all endpoints in the local Tivoli region:

```
wepstatus -a
```

The output for this command lists the endpoint label, gateway, interpreter type, endpoint version, object dispatcher number, endpoint status, and error codes, as shown in the following example:

```
Last known status of the endpoint(s):

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Gateway</th>
<th>Interp</th>
<th>Version OD</th>
<th>Status</th>
<th>Error Code(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gem_808_nt02</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>10 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_nt03</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>11 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_nt04</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>12 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_nt05</td>
<td>ntgw</td>
<td>w32-ix86</td>
<td>41100</td>
<td>13 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol06</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>14 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol07</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>15 unknown</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol08</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>16 unreachable</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol09</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>17 unreachable</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol10</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>18 unreachable</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol11</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>19 unreachable</td>
<td></td>
</tr>
<tr>
<td>gem_808_sol12</td>
<td>ntgw</td>
<td>solaris2</td>
<td>41100</td>
<td>20 unreachable</td>
<td></td>
</tr>
</tbody>
</table>

See Also

wgateway
wepupgd

Upgrades an endpoint.

Syntax

```
wepupgd -h
```

```
```

Description

The `wepupgd` command upgrades endpoints or forces the installation of a previous version of an endpoint. This command can be issued from any managed node in the Tivoli region, but it can be used to administer all supported endpoint platforms.

**Note:** This command replaces the `wadminep upgrade` and `wadminep staged_upgrade` commands.

Options

- `-a` Upgrades all endpoints in the Tivoli region.
- `-f` Forces the installation of a previous version of an endpoint.
- `-g gw_name`
  Upgrades all endpoints associated with the specified gateway.
- `-h` Displays a detailed usage statement.
- `-i file`
  The fully-qualified file name containing the list of endpoint to be upgraded. Each endpoint in the file must be on a separate line.
- `-l`
  Retrieves the upgrade package from the local managed node. When this option is not specified, retrieves the upgrade package from the gateway assigned to the specified endpoints. This option is not available on NetWare gateways.
- `-L config_options`
  Passes configuration options to the `lcfd` command for starting the endpoint. If you specify multiple options or have spaces in a single option, you must enclose the text in double quotation marks (") as follows:

  `-L "-p 3456 -P 7890"

  See the `lcfd` command for a list of valid options.

- `-p upgrade_path`
  Specifies the fully-qualified path of the upgrade package. The default location is the appropriate lcf_bundle directory.

- `-r`
  Reboots a Windows 98 endpoint after the upgrade.

- `-v`
  Runs the command in verbose mode.

- `-w` Specifies, in seconds, how long the gateway waits for the endpoint to log back in after an upgrade. The default is 150 seconds.

  `ep_name...`

  Specifies the names of the endpoints to be upgraded.
wepupgd

Authorization

The **admin** role is required with the privilege of root or Administrator on the endpoint.

Examples

1. The following example upgrades endpoint packer. The upgrade package is located in the lcf_bundle directory of the assigned gateway.
   
   weputpgd packer

2. The following example upgrades endpoints packer and hoth. The upgrade package is located in the lcf_bundle directory of the assigned gateway.

   weputpgd packer hoth

3. The following example forces the installation of a previous version of an endpoint on odin. The endpoint image is located in the data/bin/TMF37/lcf_bundle directory of the managed node where the command is issued.

   weputpgd -f -l -p /data/bin/TMF37/lcf_bundle odin

4. The following example upgrades all endpoints in the Tivoli region.

   weputpgd -a

5. The following example upgrades all endpoints assigned to gateway steeler. The upgrade package is located in the lcf_bundle directory of the managed node where the command is issued.

   weputpgd -l -g steeler

See Also

lcfd | winstlcf
wexpnotif

Expires notices from a notice group.

**Syntax**

```
wexpnotif [-a age] notice_group
```

**Description**

The `wexpnotif` command expires notices from a notice group. Notice groups typically have an associated expiration time. This command can be used to force the expiration of notices before their usual expiration time. If the `-a` option is specified, only notices older than the specified age are expired. If omitted, all notices in the specified notice group expire immediately.

**Options**

- `-a age`  Specifies that only notices older the specified number of hours are to be expired. If omitted, all notices in the specified notice group expire immediately.

- `notice_group`  Specifies the notice group for which notices are to be expired.

**Authorization**

You must have at least the `senior` role in the Tivoli region.

**Examples**

The following example specifies that all Tivoli Administration notices should be expired after 3 hours:

```
wexpnotif -a 3 "Tivoli Administration"
```

**See Also**

- `wlsnotif`
- `wsndnotif`
- `wtailnotif`
wgateway

Starts or stops a gateway, modifies or list properties of a gateway, controls access to the gateway HTTP server, or defines logging characteristics.

Syntax

wgateway

wgateway gateway_name [add_gatewayproxy node_name | get_gatewayproxies | remove_gatewayproxy node_name | reset_gatewayproxies]

wgateway gateway_name [add_protocol protocol | rm_protocol protocol] [set_ipx_port port] [set_nbi_port port] [set_nbi_name name] [set_protocols protocol_list] [set_sna_address address] [set_sna_mode mode] [set_sna_port port] [set_tcp_port port]

wgateway gateway_name [dbcheck | epact_dbcheck]

wgateway gateway_name describe

wgateway gateway_name get option

wgateway gateway_name logstatus

wgateway gateway_name [restart | start | stop]

wgateway gateway_name run option value

wgateway gateway_name set option value

Description

The wgateway command starts or stops a gateway, lists or modifies the properties of a gateway, controls access to the gateway HTTP server, or defines logging characteristics.

Without options, the wgateway command lists the object identifier, label, and status of all gateways in the local Tivoli region. The status value for a gateway can be one of the following:

b  The gateway is booting.
d  The gateway is down and not running.
D  A communication error occurred. The object dispatch is down. However, the entire computer could be down.
u  The gateway is up and running.

To start, stop, or restart (stop and start) a gateway, use the start, stop and restart options, respectively.

Note: For NetWare gateways, the start and stop options cannot be used. For the start option, use the oservrun command. For the stop option, use the oservend command. Both of these commands must be run from the NetWare console.
To list all the properties of a specific gateway, use the `describe` option. If you need to change any of these properties, you can modify many of these properties using the `set` option and its suboptions.

For endpoint method logging, use the `get method_trace_time` option to return the value of the set resolution time and the `set method_trace_time` option to change the resolution time. To return the status of logging, use the `logstatus` option. To change the amount of time between status checks, use the `set logstatus_interval` option.

To ensure that access is secure to the gateway HTTP server, you can enable or disable the server by using the `set httpd` option. After the gateway HTTP server is enabled, anyone connecting to this service is prompted to authenticate. No one can access the gateway Web pages until the user names and passwords are created. After setting the user names and passwords, access to the gateway pages for those users is allowed. User names and passwords must be predefined for each user using the `set httpd_passwd` option. Each administrator with the `admin` authorization role can define his or her user name and password.

**Note:** The user name and password can be any value. They do not default to the system administrator user name or password. They do not default to the Tivoli administrator user name and password.

For NetWare gateways, you need at least one gateway proxy. The gateway proxy runs endpoint policy scripts. Any managed node can be a gateway proxy. The following options can be used to manage gateway proxies:

`add_gatewayproxy`
To add a managed node to the gateway proxy list.

`get_gatewayproxies`
To list the gateway proxies for a specific NetWare gateway.

`remove_gatewayproxy`
To remove a managed node from the gateway proxy list.

`reset_gatewayproxies`
To remove all entries from the gateway proxy list.

For NetWare gateways and other gateways that support the IPX/SPX protocols, you can add IPX/SPX support after creating the gateway. To add IPX/SPX support, use the `add_protocol IPX` option and use the `set protocols IPX` option to set the protocols for the gateway. If you no longer need IPX/SPX support, use the `rm_protocol IPX` option. If you do not want to use the standard TCP/IP or IPX/SPX listening ports, use the `set_tcp_port` to set the TCP/IP listening port or `set_ipx_port` option to set the IPX and SPX listening ports.

Because each gateway maintains a database file and has a method cache, it could become unsynchronized with the one on the Tivoli server or become corrupted. To synchronize the method caches between these computers, use the `dbcheck` option. To check the consistency of the endpoint activity database (the epact.bdb file), use the `epact_dbcheck` option.

**Options**

`add_gatewayproxy node_name`
For NetWare gateways only. Adds a managed node as an entry to the gateway proxy list.
add_protocol protocol
   Adds a supported protocol for the specified gateway. Supported protocols are TCPIP (the default) and IPX.

dbcheck
   Synchronizes the method cache on the gateway with the method cache on the Tivoli server.

describe
   Lists the properties of the specified gateway.

epact_dbcheck
   Checks the consistency of the endpoint activity database (the epact.bdb file).

get option
   Returns the value that is set for the specified option. You can specify the following suboptions with the get option:

   continue_eplogin_onerror
      Returns a value indicating whether the gateway allows exceptions from the boot_method method, the login_policy script, or both, during normal login attempts by endpoints.

debug_level
   Returns the level of message information that is logged by the gateway. The levels are as follows:

epcheck_atboot
   Returns a value indicating whether the gateway checks the status of all endpoints that are assigned to it each time the gateway starts up.

epcheck_interval
   Returns the interval, in seconds, at which the gateway checks the status of endpoints that it communicates with.

eplogin_timeout
   Returns the current value of the endpoint login timeout.

epupgrade_by_gateway
   Returns a value indicating whether the gateway upgrades any endpoints assigned to it during the normal login process. If the endpoint is already at the most current level, no upgrade is performed.

httpd
   Returns a value indicating whether the gateway HTTP server is enabled.

httpd_passwd
   Displays the user name and password required to access the gateway HTTP server to view information about endpoints.

ifs_ignore
   Displays a list of IP addresses that are excluded from the gateway aliases list when the gateway boots up.

ipx_port
   Returns the port on which the gateway listens for IPX packets.

log_dir
   Returns the directory on the local system for the gatelog and all gatelog.timestamp log files.
log_size
Returns the reference size of the gatelog file, in bytes.

logstatus_interval
Returns the amount of time in seconds between status checks.

max_concurrent_jobs
Returns the maximum number of jobs that run concurrently.

max_concurrent_logins
Returns the maximum number of endpoints that can log in at the same time.

mcache_bwcontrol
Returns a value that indicates whether the gateway shares bandwidth control between the MDist 2 service and the method cache download.

method_trace_time
Returns the resolution time in seconds for logging of endpoint methods.

nbi_name
Returns the NetBIOS name.

nbi_mode
Returns the NetBIOS name resolution mode.

nbi_port
Returns the port on which the gateway listens for NetBIOS packets.

protocols
Returns the supported protocol for the specified gateway.

rpc_maxthreads
Returns the maximum number of threads.

send_userlink_file
Returns a value that indicates whether the gateway is enabled to send the userlink.htm file to an endpoint.

session_timeout
Returns the amount of time in seconds that a gateway waits for a response from an endpoint after sending a downcall.

sna_address
Returns the SNA address.

sna_mode
Returns the SNA mode.

sna_port
Returns the port on which the gateway listens for SNA packets.

tcp_port
Returns the port on which gateway listens for TCP packets.

wol_broadcast
Returns a value that indicates whether the gateway broadcasts a Wake on Lan packet to the subnet of the endpoint.

wol_direct
Returns a value that indicates whether the gateway sends a Wake on Lan packet directly to the IP address of the endpoint.
wgateway

get_gatewayproxies
   For NetWare gateways only. Displays the list of managed nodes that have been added to the named gateway.

logstatus
   Returns logging status.

remove_gatewayproxy node_name
   For NetWare gateways only. Removes a managed node as an entry from the gateway proxy list.

reset_gatewayproxies
   For NetWare gateways only. Clears the gateway proxy list.

restart
   Stops and restarts the specified gateway.

rm_protocol IPX
   Removes support of the IPX protocol from the specified gateway. The TCP/IP protocol cannot be removed.

run
   Runs the specified suboption. You can specify the following suboptions with the run option:

   sync_login_interval
      Synchronizes the endpoint login interval for the gateway with the endpoint manager value. You do not have to reboot the gateway for this change to take effect.

   ifs_ignore_remove IP_address
      Removes the specified IP address from the list of excluded gateway aliases.

set option value
   Sets the value for the specified option. You can specify the following options and values with the set option:

   continue_eplogin_onerror {0 | 1 | 2 | 3}
      Specifies whether the gateway allows exceptions from the boot_method method, the login_policy script, or both, during normal login attempts by endpoints. Specify one of the following values:

      0  The gateway aborts the login attempt if receives an exception from the boot_method method or login_policy script. This is the default value.

      1  The gateway allows exceptions from the login_policy script.

      2  The gateway allows exceptions from the boot_method method.

      3  The gateway allows exceptions from both the login_policy script and the boot_method method.

   debug_level level
      Sets the level of message information that is logged by the gateway. The levels are as follows:

      0  Errors. This is the default and recommended level.

      1  Errors and warnings.

      2  Harmless exceptions.
Verbose communication information.

Verbose boot, database check, and endpoint login information.

Verbose upcall, downcall, and repeater information.

Verbose job scheduler information.

Verbose gateway cache information.

Note: Level 4 does not exist.

epcheck_atboot [TRUE | FALSE]
Specifies whether the gateway checks the status of all endpoints assigned to it each time the gateway starts up. The default is TRUE.

epcheck_interval seconds
Specifies the interval, in seconds, at which the gateway checks the status of endpoints that it communicates with. The range for this option is 300 to 14400 seconds. The default is 3600 seconds (1 hour).

epupgrade_by_gateway [TRUE | FALSE]
Upgrades any endpoint assigned to this gateway during the normal login process. If the endpoint is already at the most current level, no upgrade is performed. The default for all gateways, except NetWare gateways, is FALSE. For NetWare gateways, this value is set to TRUE. A NetWare gateway cannot run a login_policy script without first defining a proxy managed node.

Note: The epupgrade_by_gateway option works only if the upgrade_mode parameter in the upgrade.cntl file is set to auto. The upgrade.cntl file is located in the $BINDIR/lcf_bundle.41100/upgrade directory on the gateway.

httpd [TRUE | FALSE]
Enables and disables the gateway HTTP server. The option requires only the admin role. TRUE enables the gateway HTTP server; FALSE disables the gateway HTTP server. The default is FALSE.

httpd_passwd user_name:password
Sets the user name and password required to access the gateway HTTP server to view information about endpoints. This option requires only the admin role.

ifs_ignore IP_address
Specifies an IP address that is excluded from the gateway aliases list when the gateway boots up. Use this option on multi-NIC gateways to specify network cards that you do not want endpoints to connect to. You can specify only one IP address each time you run this command. However, you can run this command multiple times to exclude multiple IP addresses.

log_dir directory
Specifies the directory on the local system for the gatelog and all gatelog.timestamp log files.

log_size size_in_bytes
Specifies the reference size of the gatelog file, in bytes. When the referenced size is reached, the gatelog file is copied to a
 wgateway

gatelog.timestamp file and a new gatelog file is created. The maximum file size is 10485760 bytes (10 megabytes), and the default file size is 1048576 bytes (1 megabyte).

logstatus_interval seconds
Sets the amount of time in seconds between status checks. If set to zero, logging is disabled. The range for this option is zero to 3600 seconds. The default is 3600 seconds (1 hour).

max_concurrent_jobs count
Specifies the maximum number of jobs that run concurrently. The range for this option is 200 to 2000 jobs. The default is 200.

max_concurrent_logins count
Specifies the maximum number of endpoints that can log in at the same time. The range for this option is 100 to 500 endpoints. The default is 100.

mcache_bwcontrol [TRUE | FALSE]
Specifies whether to share bandwidth control between the MDist 2 service and the method cache download. The default is FALSE.

method_trace_time seconds
Specifies the resolution time in seconds for tracing endpoint activity (last login time, last logout time, last migration time, last downcall time, last upcall time, and last endpoint check time). When this value is set to zero, endpoint tracing information is not written to the epactbdb database but is stored in memory on the gateway. The default is 3600 seconds (1 hour).

protocols protocol_list
Sets a supported protocol for the specified gateway. TCP/IP is the default.

rpc_maxthreads count
Specifies the maximum number of threads. The range for this option is 250 to 2500 threads. The default is 250.

send_userlink_file [TRUE | FALSE]
Specifies whether the gateway sends the userlink.htm file to an endpoint. When this value is set to TRUE, the gateway sends the userlink.htm file to an endpoint after the endpoint completes its first successful normal login. The userlink.htm file is stored on the endpoint in the /etc/Tivoli directory on UNIX systems and the c:/etc/Tivoli directory on Windows systems. When this value is set to FALSE, the gateway does not send the userlink.htm file to an endpoint. The default is TRUE.

Note: This option is designed for new endpoints. The userlink.htm file might already be installed on existing endpoints.

session_timeout seconds
Specifies the amount of time in seconds that a gateway waits for a response from an endpoint after sending a downcall. The default is 300 seconds (5 minutes).

wol_broadcast [TRUE | FALSE]
Specifies whether the gateway broadcasts a Wake on Lan packet to the subnet of the endpoint. In broadcast mode, if the IP address of
an endpoint is unknown, the gateway can wake up the endpoint by sending a Wake on Lan packet to the subnet of the endpoint. The default is TRUE.

**wol_direct** [TRUE | FALSE]
Specifies whether the gateway sends a Wake on Lan packet directly to the IP address of the endpoint. The default is FALSE.

**set_ipx_port** **port**
Specifies the port on which the gateway listens for IPX packets. When an IPX port is assigned, an SPX port is assigned. The SPX port is always the assigned IPX port number less one. The default IPX port number is 9494, and the default SPX port is 9493.

**set_nbi_name** **name**
Specifies the NetBIOS name.

**set_nbi_mode** **mode**
Specifies the NetBIOS name resolution mode.

**set_nbi_port** **port**
Specifies the port on which the gateway listens for NetBIOS packets.

**set_protocols** **protocol_list**
Sets the supported protocol for the specified gateway. Supported protocols are TCPIP (the default) and IPX.

**set_sna_address** **address**
Specifies the SNA address.

**set_sna_mode** **mode**
 Defines the SNA mode.

**set_sna_port** **port**
Specifies the port on which the gateway listens for SNA packets.

**set_tcp_port** **port**
Specifies the port on which the gateway listens for TCP/IP packets. The default port number is 9494.

**start**
Starts the specified gateway. For NetWare, use the oservrun command to start the object dispatcher from the NetWare console.

**stop**
Stops the specified gateway. For NetWare, use the oservend command to stop the object dispatcher from the NetWare console.

**gateway_name**
Identifies the name of the gateway.

**Authorization**
admin or senior

**Examples**

1. The following example displays the results of the *wgateway* command when the gateway uses IPX and TCP/IP protocols:

```
wgateway solgw describe
```

```
Gateway Object : 1094978554.1.579#TMF_Gateway::Gateway#
Hostname : packer
debug_level : 9
session_timeout : 30
Protocols : TCPIP, IPX
```
### wgateway

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPIP Port</td>
<td>41684</td>
</tr>
<tr>
<td>IPX Port</td>
<td>41690</td>
</tr>
<tr>
<td>Start Time</td>
<td>2003/10/24 11:26:05 +06</td>
</tr>
<tr>
<td>no_endpoints</td>
<td>2</td>
</tr>
<tr>
<td>log_dir</td>
<td>/data/gem/1013/packer.db</td>
</tr>
<tr>
<td>log_size</td>
<td>1024000</td>
</tr>
<tr>
<td>rpc_maxthreads</td>
<td>350</td>
</tr>
<tr>
<td>max_concurrent_jobs</td>
<td>200</td>
</tr>
<tr>
<td>max_concurrent_logins</td>
<td>200</td>
</tr>
<tr>
<td>method_trace_time</td>
<td>1800</td>
</tr>
<tr>
<td>logstatus_interval</td>
<td>1800</td>
</tr>
<tr>
<td>login_interval</td>
<td>0</td>
</tr>
<tr>
<td>epcheck_interval</td>
<td>300</td>
</tr>
<tr>
<td>epcheck_atboot</td>
<td>Enabled</td>
</tr>
<tr>
<td>httpd</td>
<td>Enabled</td>
</tr>
<tr>
<td>mcache_bwcontrol</td>
<td>Disabled</td>
</tr>
<tr>
<td>wol_broadcast</td>
<td>Disabled</td>
</tr>
<tr>
<td>wol_direct</td>
<td>Enabled</td>
</tr>
<tr>
<td>send_userlink_file</td>
<td>Disabled</td>
</tr>
<tr>
<td>crypt_mode</td>
<td>NONE</td>
</tr>
<tr>
<td>ifs_ignore</td>
<td>NIL</td>
</tr>
</tbody>
</table>

2. The following example displays the results of the `wgateway` command when the gateway uses the TCP/IP protocol:

```bash
wgateway liliana-gateway describe
```

Gateway Object : 1849216852.1.580#TMF_Gateway::Gateway#
Hostname: liliana
debug_level: 9
session_timeout: 300
Protocols: TCPIP
TCPIP Port: 21234
Start Time: 2003/09/26 16:36:03 +06
no_endpoints: 3
log_dir: /data/lorozco/Tivoli-1234/db/liliana.db
log_size: 1024000
rpc_maxthreads: 350
max_concurrent_jobs: 200
max_concurrent_logins: 100
method_trace_time: 3600
logstatus_interval: 3600
login_interval: 270
epcheck_interval: 3600
epcheck_atboot: Enabled
gateway_httpd: Disabled
mcache_bwcontrol: Disabled
gateway_crypt_mode: NONE
wol_broadcast: Enabled
wol_direct: Disabled
Interface Ignore List: None
send_userlink_file: Enabled

3. The following example adds managed node lradner as an entry to the gateway proxy list of the NetWare gateway lux:

```bash
wgateway lux add_gatewayproxy lradner
```

4. The following example displays the gateway proxy list of the NetWare gateway lux:

```bash
wgateway lux get_gatewayproxies
```

5. The following example removes the specified IP address from the list of excluded aliases of the gateway lux:

```bash
wgateway lux run ifs_ignore_remove 9.99.99.98
```

6. The following example specifies that a maximum of 200 endpoints can log in to gateway lux at the same time.

```bash
wgateway lux set max_concurrent_logins 200
```
See Also

wcrtgate  wdelgate  wrpt
wgetadmin

Lists information about a Tivoli administrator.

Syntax

wgetadmin [name]
wgetadmin -n [name]
wgetadmin -o [name]
wgetadmin -p [name]
wgetadmin -u [-i interp] [name]

Description

The wgetadmin command lists information about a Tivoli administrator to standard output. This command lists the name, logins, roles, and notice groups for an administrator. The administrator roles are listed by groups. If name is not specified, the command lists the information about the current administrator.

Note: This command does not show the user name or group name associated with the administrator. You must use the Tivoli desktop to perform this operation.

Options

- -i interp
    Resolves the user login name and group name of the administrator using the specified interpreter type.
- -n
    Displays only the name of the administrator.
- -o
    Displays only the object ID of the administrator.
- -p
    Displays extended output for multiple notice groups. Appends the Tivoli region name corresponding to the notice groups listed. This is helpful in an interconnected Tivoli environment.
- -u
    Displays the user login name and the group name of the administrator.

name
    Specifies the name of the Tivoli administrator whose properties to list.

Authorization

You must have user in the Tivoli region.

Examples

1. The following example displays administration information about the current administrator:
   wgetadmin
2. The following example displays administration information about administrator callahan@sthelens:
   wgetadmin callahan@sthelens
   Administrator Steve Callahan
   logins: callahan@sthelens
roles: global user
  DefaultRegion super, admin, user
  Administrators super, senior
  MyRegion super, senior, admin, user, backup
  Steve Callahan admin, user, rconnect
  security_group_any_admin user
notice groups: Tivoli Authorization

See Also
  wgetadmin  widmap  wsetadmin
wgetallinst

Displays all instances of a resource type.

Syntax
wgetallinst [–I] resource_type

Description
The wgetallinst command displays all instances of a resource type. If the –I option is included, this command displays the resource type in the form 'label oid'.

The wgetallinst command is similar to the wlookup command. The difference is that the wlookup command displays only those resource types registered in the name registry, while the wgetallinst command displays both registered and unregistered resource types. Unregistered resources include those resource types that are supersets of other resource types. For example, the wgetallinst command displays all instances of the ProfileEndpoint resource type, which includes instances of the ProfileManager, ManagedNode, and NisDomain resource types.

Options
–I     Lists the instances in the form 'label oid'.

resource_type
     Specifies the resource type of the instances to be displayed.

Examples
The following example displays all instances of the ProfileEndpoint resource type:
wgetallinst -l ProfileEndpoint

See Also
wlookup
wgetdfpol

Lists a default policy object.

Syntax

wgetdfpol [-d | -v] class

Description

The wgetdfpol command lists the label of the default policy default or policy validation object for the resource with the specified label. When a resource is added to a policy region as a managed resource, it receives both a default policy validation object and a default policy default object. These defaults are predefined as part of the resource definition. A policy default object generates default values when creating objects of a given resource type in a policy region. A policy validation object implements the checking of attribute values for the objects of a managed resource type in a policy region.

Options

- `-d` Lists the labels of the default policy default objects. This option is the default if `-v` is not specified.
- `-v` Lists the labels of the default policy validation objects.

`class` Specifies the managed resource class for which to retrieve the default policy objects.

Authorization

`senior` or `super`

Examples

The following example returns the name of the default policy default object for the ProfileManager class.

```
wgetdfpol -d ProfileManager
```

See Also

wchkpol, wcrtpol, wcrtpr, wdelpol, wdelpr, wgetpolm, wlspl, wlsplm, wputpolm
wgeteppol

Lists the body and constant values of an endpoint policy script.

Syntax

wgeteppol policy_name

Description

The wgeteppol command lists the contents of the specified endpoint policy script. You can modify the script to meet the needs of your organization. The endpoint policy scripts are allow_install_policy, after_install_policy, login_policy, and select_gateway_policy.

If you have not added an endpoint policy script yet, the output of wgeteppol will be the shell of a script. Add the contents of the policy script after the comments and before the exit statement. Then use the wputeppol command to write the new script to disk.

Options

policy_name

Specifies the name of the policy script to be returned.

Authorization

senior

Examples

The following example returns the after_install_policy script, which can then be modified as needed:

wgeteppol after_install_policy
#!/bin/sh
#
# The following are the command line options passed to
# this script from the Endpoint Manager.
#
# $1 - The label of the endpoint machine
# $2 - The object reference of the endpoint machine
# $3 - The architecture type of the endpoint machine
# $4 - The object reference of the gateway that the
# endpoint logged into
# $5 - The ip address of the endpoint logging in.
# $6 - Region
# $7 - Dispatcher
# $8 - Version
# $9 - The inventory id of the endpoint logging in.
# The following command line option will be passed to this script
# from the Endpoint Manager, when complied with the MULTIPROTO flag
# turned on
# $10 - The protocol of the endpoint logging in.
# TCPIP -> TCP/IP
# IPX -> IPX/SPX
# Note that the environment variable LCF_LOGIN_STATUS is also set by
# the endpoint manager. A value of 2 indicates the endpoint
# is isolated. That is, it was unable to contact its
# assigned gateway. Isolated endpoints are automatically
# migrated to another gateway unless the
# select_gateway_policy terminates with a nonzero exit
# status: #
#Also note that during the execution of
#allow_install and select_gateway policy scripts,
#the endpoint does not yet formally exist. For this
#reason, the endpoint object reference will have a
#value of OBJECT_NIL, and the object dispatcher number
#will be 0. The endpoint label will have the value
#suggested by the endpoint (or the user value lcfd -n),
#but is not guaranteed to become the final endpoint
#label. It will become the final endpoint label if
#this value is not already taken by another endpoint.
exit 0
#

See Also

wgeteppol
wgetjob

Lists the properties of a job.

Syntax

```
wgetjob task_name library_name
```

Description

The `wgetjob` command lists the properties of a job to standard output.

Options

```
library_name
```

Specifies the name of the task library containing the specified job.

```
task_name
```

Specifies the name of the job to be listed.

Authorization

```
user, admin, senior, super
```

Examples

The following example lists the properties for the Clean Queue job:

```
wgetjob "Clean Queue" queue_lib
```

```
Job Name : Clean Queue
Task Name : Clean Queue
Execution Mode : parallel
Timeout : 60
Output Format : task header
               return code
               task standard output
               task standard error

Managed Nodes : yogi

Profile Managers : 
```

See Also

```
wcrtjob, wdeljob
```
wgetkey

Retrieves the subkey listing in a registry hive. This command should be run from an endpoint. (Windows only)

Syntax

`wgetkey registry_key_path [registry_entry]`

Description

The `wgetkey` command retrieves the subkeys associated with the specified key path from the specified registry entry. The list is returned to standard output.

Options

`registry_entry`

Specifies the registry entry from which to retrieve the subkeys. Valid entries are as follows:

- HKEY_LOCAL_MACHINE
- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_USERS

If you do not specify this option, the command retrieves the subkeys from the HKEY_LOCAL_MACHINE entry.

`registry_key_path`

Specifies a registry key name from which to retrieve the subkeys.

Authorization

admin

Examples

1. The following example retrieves all of the subkeys from the HKEY_LOCAL_MACHINE entry under the SOFTWARE key path:

   `wgetkey SOFTWARE`

2. The following example retrieves the key values from the HKEY_CURRENT_USER entry under the USERS key path:

   `wgetkey USERS HKEY_CURRENT_USER`
wgetpolm

Lists the body or constant value of a default or validation policy method.

Syntax

wgetpolm –d class name policy
wgetpolm –v class name policy
wgetpolm –d profile policy
wgetpolm –v profile policy

Description

The wgetpolm command lists the body or constant value of a policy method to standard output. This command returns one of two things. If the policy method is implemented as a shell script (or executable program), this method returns the body of the script (or program). If the policy method is implemented as a constant value, it returns that constant value. This command does not explicitly indicate which way the method is implemented.

Note: For policy methods that are executable programs, this command returns the binary image of the executable program.

The –d option (default) returns a default policy method, and –v returns a validation policy method. The class and name options specify the managed resource type and policy object name, respectively, while the policy option specifies the individual attribute whose policy method to return, in the same form as the wlspolm command.

The second form of the command is used to query a policy from a profile. In this case, profile specifies the profile to query, and policy specifies the individual attribute whose policy to return, again as returned by the wlspolm command. If the policy is implemented as a constant value, this command returns that value. If the policy is implemented as a shell script (or executable program), it returns the script (or program) body to standard output, and prints the options to the script to standard error (see the wputpolm command). If the policy is undefined or specified as none, a message to that effect is printed to standard error and nothing is printed to standard output.

Options

–d  Lists the specified policy default method. This action is the default if the –v option is not specified.
–v  Lists the specified policy validation method.
class  Identifies the managed resource type whose policy is to be returned.
name  Identifies the name of the policy object.
policy  Identifies the attribute whose policy to return.
profile  Identifies the profile whose policy to return.
Authorization

You must have at least the senior role.

Examples

For nonprofile usage, the following example returns the script body for the pm_val_subscribers method of the BasicProfileManager policy validation object of the ProfileManager resource:

```
wgetpolm -v "ProfileManager" "BasicProfileManager" "pm_val_subscribers"
```

For profile usage, the following example returns the script body for the default user ID (UID) policy for the Engineering profile named.

```
wgetpolm -d @UserProfile:user_profile_2 uid
```

script ARGUMENTS: $real_name $login_name

See Also

wchkpol | wcrtpol | wcrtpr | wdelpol | wdelpri | wgetdfpol | wspolm | wputpolm
wgetpr

Lists the properties of a policy region.

Syntax

```
wgetpr region
```

Description

The `wgetpr` command lists the properties of a policy region.

Options

```
region  Specifies the target policy region.
```

Authorization

```
admin, senior, super
```

Examples

The following example lists all the managed resources in the DefaultRegion policy region:

```
wgetpr @PolicyRegion:DefaultRegion
```

TaskLibrary
ManagedNode
ProfileManager

See Also

`wsetpr`
wgetprf

Retrieves subscription copies of one or more profiles.

Syntax

wgetprf [-l maintain | over_all] [-m] [-r] name

Description

The wgetprf command retrieves subscription copies of one or more profiles from
the profile managers to which the current profile manager or endpoint (host or
Network Information Services [NIS] domain) subscribes. This command updates
the subscriber databases and configuration files from the Tivoli database. The name
option specifies an endpoint to which to send the copy. It can also specify the
current profile copy.

If the –m option is specified, a multistep distribution is performed from the profile
manager. If omitted, a single-step distribution is performed.

The –l option identifies the distribution level. If omitted, defaults to maintain.

Options

–l maintain | over_all
Specifies the distribution level. The maintain option keeps local
modifications, while the over_all option overwrites local modifications. The
default is maintain.

–m
Specifies a multistep distribution.

–r
Sets the return code to 1 if at least one profile distribution to or retrieval
from a profile manager fails.

name
Specifies the endpoint to which to send the copy, or specifies a current
profile copy. Valid formats are as follows:
• @node_name
• @ManagedNode:node_name
• /Regions/policy_region_name/node_name

Authorization

admin, senior, super

Examples

1. The following example retrieves the profile copy Users, overwriting any local
modifications:
wgetprf -l over_all @UserProfile:Users@rushmore

2. The following example retrieves all profile copies in managed node rushmore
recursively, maintaining any local modification:
wgetprf -m @ManagedNode:rushmore

See Also

wcrpr, wcrprfmg, wdistrib, wgetsub, wlssub, wpopulate, wsub, wunsub
wgetquery

Lists information about a query.

Syntax

wgetquery [-f] query_name

Description

The wgetquery command lists information about a Tivoli query. This information includes the name, description, repository, view, columns list, and where clause.

Options

-f Lists all information about the specified query. When omitted, lists only the where clause.

query_name Specifies the name of the query.

Authorization

query_view, user, admin, senior, or super

Examples

The following example lists all information about the DOS-machines query:

wgetquery -f DOS-machines

Name: DOS-machines
Description: Query for DOS PCs
Repository: inventory
View: MACHINE_TYPE
Columns: PROCESSOR_TYPE
        OPERATING_SYSTEM

Where Clause:

--------------------
(BOOTED_OS_NAME = 'DOS')

See Also

wcrtqlib wcrtquery wruninvquery wsetquery
**wgetrim**

Lists information about an RDBMS Interface Module (RIM) object.

**Syntax**

```
wgetrim rim_name
```

**Description**

The `wgetrim` command lists the current information for the specified RIM object. This command returns all configuration information except the values for application label and maximum connections. To obtain this information, use the appropriate invocation of the `idlcall` command.

**Options**

`rim_name`

Specifies the label of the RIM object. This can be either the Tivoli name registry label (in the form `@RIM:name`) or the text label. To ensure that you always find the RIM object, precede `rim_name` with `@` character.

**Authorization**

`senior` or `super` in the Tivoli region

**Examples**

1. The following example returns the information for the inventory RIM object:

   ```
   wgettrim invdh_1
   ```

   The output is as follows:
   
   - **RIM Host:** fwksrv1
   - **RDBMS User:** invtiv
   - **RDBMS Vendor:** DB2
   - **Database ID:** invdb
   - **Database Home:** /usr/lpp/db2_07_01
   - **Server ID:** tcpip
   - **Instance Home:** ~invtiv
   - **Instance Name:** invtiv

2. To determine the application label and maximum connection for a RIM object, perform the following steps:

   a. List all RIM object in the Tivoli environment by object identifier (OID):
      ```
      wlookup -ar RIM
      ```

   b. List the application label:
      ```
      idlcall OID _get_application_type
      ```

   c. List the maximum number of connections:
      ```
      idlcall OID _get_max_conn
      ```

**See Also**

`wcrtrim`, `wsetrim`, `wsetrimpw`
wgetsched

Retrieves information about jobs currently scheduled.

Syntax

wgetsched [-b 'mm/dd/yyyy hh:mm'] [-a 'mm/dd/yyyy hh:mm'] [-v]
wgetsched [-s id...] [-v]

Description

The wgetsched command retrieves information about jobs currently scheduled to execute. If no options are specified, information about all jobs is displayed.

Options

-a 'mm/dd/yyyy hh:mm'
   Specifies the starting time for the search.

-b 'mm/dd/yyyy hh:mm'
   Specifies the ending time for the search.

-s id...
   Specifies the job ID. You can specify more than one job ID.

-v
   Specifies verbose mode.

Authorization

user

Examples

1. The following example lists scheduling information for jobs 2, 8, and 10:

   wgetsched -s 2 -s 8 -s 10

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Label</th>
<th>Admin</th>
<th>Date &amp; Time</th>
<th>Enbld</th>
<th>Repeat</th>
<th>Retry</th>
<th>Cncl</th>
</tr>
</thead>
<tbody>
<tr>
<td>000008</td>
<td>JOB_14</td>
<td>root@cook</td>
<td>Fri May 6 01:00:00 2003</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>000002</td>
<td>JOB_2</td>
<td>root@cook</td>
<td>Sun Jan 1 01:12:00 2004</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>000010</td>
<td>JOB_9</td>
<td>root@cook</td>
<td>Wed Mar 1 06:55:00 2004</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

2. The following example lists verbose output for the job 2:

   wgetsched -s 2 -v

   ID         : 2
   Name       : Backup
   Label      : Backup
   Description:
   Administrator: root@cook
   Original Time: Tue Mar 05 16:00:00 2003
   Next Time  : Tue Mar 05 16:00:00 2003
   Enabled    : Yes
   Repeat Type: Infinite
   Repeat Increment: 1
   Repeat Unit : Day
   Repeat Times : 0
   Retry Type  : None
   Retry Increment: 0
   Retry Unit  : Minute
   Retry Times : 0
   Cancel Job : Yes
   Cancel Increment: 10
   Cancel Unit : Minute
Email
Notice : Tivoli Scheduler
Desktop :
Host Name :
File Name :
Daytime Rest. : No
Daytime From : 6
Daytime To : 18
Nighttime Rest. : No
Nighttime From : 17
Nighttime To : 8
Weekday Rest. : No
Weekday From : 1
Weekday To : 5
Weekend Rest. : Yes
Weekend From : 6
Weekend To : 0

3. The following example lists the jobs scheduled to start before 1:00 a.m. on May 6, 1998 and after 1:00 p.m. on May 8, 1998:

wgetsched -b '05/06/1998 01:00' -a '05/08/1998 13:00'

4. The following example lists the jobs scheduled to start after midnight on August 10, 1998 and before 6:00 p.m. on August 12, 1998:

wgetsched -a '08/10/1998 00:00' -b '08/12/1998 18:00'

See Also

wdelsched | wedsched | wenblsched | wschedjob
wgetsub

Lists subscribers to a profile manager.

**Syntax**

```
wgetsub [-l] [-o] name
```

**Description**

The `wgetsub` command lists the subscribers to a specific profile manager.

**Options**

- `-l` Specifies a long listing.
- `-o` Lists the object identifier for each subscriber.
- `name` Specifies the name of the profile manager containing the subscribers. Valid formats are as follows:
  - `@profile_manager_name`
  - `@ProfileManager:profile_manager_name`
  - `/Regions/policy_region_name/profile_manager_name`

**Authorization**

user, admin, senior, super

**Examples**

The following example lists all the subscribers of profile manager Development:

```
wgetsub @Development
```

**See Also**

`wcrtprf` `wcrtprfmg` `wdistrib` `wgetprf` `wlssub` `wpopulate` `wsup` `wunsub` `wvalidate`
wgettext

Lists the properties of a task.

Syntax

wgettext [-F file_name] task_name library_name

Description

The `wgettext` command lists the properties of a task, and sends it to standard output.

Options

- `-F file_name`
  Specifies a file in which the task information is written. This option creates a .tar file of the task binaries and comments as well as a description of the task settings. This option is useful when exporting a task from one Tivoli environment to another.

- `library_name`
  Specifies the name of the task library containing the specified task.

- `task_name`
  Specifies the name of the task to be listed.

Authorization

user, senior, super

Examples

The following example lists all information about the Clean Queue task in the queue_lib task library:

```
wgettask "Clean Queue" queue_lib
```

```
Task Name : Clean Queue
User ID : *
Group ID : 
Task ACL : admin:senior:super:user
Supported Platforms :
  solaris2 <install>/solaris2/TAS/TASK_LIBRARY/bin/200000/tasknpzmqd
Task Documentation :
  Task Name : Clean Queue
  Task Created By : root@yogi
  Comments:
```

See Also

`wcrtask`, `wdeltask`
wgetval

Retrieves a registry subkey. (Windows only)

Syntax

```
wgetval [-h registry_hive] -k [key | @file_name] -n value_name
```

Description

The `wgetval` command retrieves a subkey from a registry. The output of this command is returned to standard output.

Options

```
-h registry_hive
```

Specifies the registry hive from which to retrieve the subkey. Valid values are as follows:

- HKEY_LOCAL_MACHINE
- HKEY_CURRENT_USER
- HKEY_CLASSES_ROOT
- HKEY_USERS
- HKEY_CURRENT_CONFIG
- HKEY_DYN_DATA

```
-k key | @file_name
```

Specifies the key or file from which the subkey is retrieved.

```
-n value_name
```

Specifies the name of the value.

Authorization

admin

Examples

The following example retrieves the version number of the Novell drivers:

```
wgetval -h HKEY_LOCAL_MACHINE -k SOFTWARE\NOVELL -n CurrentVersion
```

See Also

`wsetval`
whostid

Prints the host ID of the specified managed node.

**Syntax**

```
whostid host_name
```

**Description**

The `whostid` command prints the numeric host ID of the specified managed node to standard output.

**Options**

```
host_name
```

Specifies the name of the host whose ID to list.

**Authorization**

user, admin, senior, super

**Examples**

The following example lists the host ID of managed node bald:

```
whostid bald
```

```
8031ee30
```

**See Also**

`wdate`, `wdiskspace`, `wifconfig`, `winstdir`, `winterp`, `wmannode`, `wmemsiz`, `wping`, `wtimezone`, `wuname`, `wxterm`
**whttpd**

Uninstalls or unregisters the Tivoli HTTP service (spider) and optionally forwards HTTP request to a third-party Web server.

**Syntax**

```
  whttpd -f url
  whttpd -g
  whttpd -r node_name [-f url]
  whttpd -u node_name [-f url]
```

**Description**

The `whttpd` command enables you to uninstall or unregister spider and optionally to forward HTTP requests to a third-party Web server. The spider service is associated with the object dispatcher on managed nodes. If you unregister spider instead of uninstalling it, the spider binaries remain on the managed node.

**Note:** The spider service does not affect the HTTP services associated with gateway-listings or endpoint-listings.

**Options**

- `-f url`  
  Forwards HTTP requests to the specified Universal Resource Location (URL). The format of `url` is as follows:

  `protocol://web_server_address:port/TivoliFRW/webapp`

  where:

  `protocol`
  The protocol used for HTTP requests.

  `web_server_address`
  The WWW-known name or IP address of the machine hosting the third-party Web server.

  `port`
  The correctly configured HTTP port for the third-party Web server.

  You must have the `admin` role to use this option.

- `-g`
  Returns the URL of the third-party Web server. If your environment is still using spider or if you have not enabled forwarding, this command returns the message, "Forwarding URL has not been set."

  You must have the `user` role to use this option.

- `-r node_name`
  Unregisters spider and removes the spider file. If `node_name` is the Tivoli server, the command uninstalls spider on all managed nodes in the Tivoli region. You must have the `super` role to use this option.

- `-u node_name`
  Unregisters spider, but retains the spider file. If `node_name` is the Tivoli server, the command unregisters spider on all managed nodes in the Tivoli region. You must have the `super` role to use this option.
Note: Removing the spider binaries is permanent.

Examples

1. The following example unregisters spider from managed node ccorley but retains the spider file:
   
   `whttpd -u ccorley`

2. The following example forwards HTTP requests to the third-party Web server on workstation lorozco:

   `whttpd -f http://w3.lorozco.tivoli.com/TivoliFRW/webapp`
**wiconv**

Converts the characters or sequences of characters in a file from one code set to another code set.

**Syntax**

```bash
wiconv [-f code_set] [-t code_set < input]
```

```bash
wiconv [-f code_set] [-t code_set > output]
```

```bash
wiconv [-f code_set] [-t code_set] [-i input] [-o output]
```

**Description**

The `wiconv` command converts the characters or sequences of characters in a file from one code set to another code set and then writes the results to standard output. Before using this command it is necessary to set the TISDIR environment variable.

**Options**

- `-f code_set`
  Identifies the input code set.

- `-i input`
  Identifies the name of the input file instead of using the standard input.

- `-o output`
  Identifies the name of the output file instead of using the standard output.

- `-t code_set`
  Identifies the output code set.

- `> output`
  Writes the results to standard output.

- `< input`
  Reads input data from standard input.

**Authorization**

This command does not require any Tivoli region authorization roles to be executed.

**Examples**

The following example takes the data in the `source.txt` file, converts it from SJIS encoding to UTF8 encoding, and then outputs the result to the `utf8.html` file:

```bash
wiconv -f SJIS -t UTF8 -i source.txt -o utf8.html
```
wident

Identifies files containing a specified pattern.

Syntax

wident [-q] [file...]

Description

The wident command searches for all occurrences of the pattern "$keyword:...$" in the named file. If no file name is specified, then standard input.

These patterns are typically inserted automatically by the wco command, but you can insert them manually. By using the -q option, you can suppress the warning stating that the file does not contain the specified pattern.

The wident command works on text files, object files, and dumps. For example, if the C program in f.c contains the following:

```c
char rcsid[] = "$Id: f.c,v *(id $;
```

and f.c is compiled into f.o, the command:

wident f.c f.o

The outputs is as follows:

```c
f.c: $Id: f.c,v *(id $
```

```c
f.o: $Id: f.c,v *(id $
```

Author


See Also

widmap

Lists and modifies user login mappings.

Syntax

widmap add_entry map_name interp entry_val
widmap add_map map_name
widmap list_entries map_name
widmap list_maps
widmap resolve_entry map_name interp
widmap rm_entry map_name interp
widmap rm_map map_name

Description

The widmap command creates, lists, adds, and deletes entire maps or entries of a map. Using this command enables you to create and maintain mappings of user logins across multiple platforms. A login map enables Tivoli Management Framework to associate a single user login name with the correct user account on a specified operating system. For example, the login name chris might be mapped to user name chriss on Solaris operating environments and to user name chris_sanders on Windows operating systems.

The following example shows two login mappings, one for root_user and one for chris:

root_user     default       root
root_user     w32-ix86      Administrator
chris         solaris2      chriss
chris         w32-ix86      chris_sanders

Map names can be provided in Tivoli windows as $map_name. The map name, for example $chris, can be entered into the Login Name or Group Name fields of the Create Administrator window. During process, this map name is resolved to the correct user name depending on the operating system.

Options

add_entry
   Adds an map entry.
add_map
   Adds a map.
list_entries
   Lists the existing entries for a map.
list_maps
   Lists the existing maps.
resolve_entry
   Returns the value of entry_val based on interpreter type.
**rm_entry**
Removes a map entry based on interpreter type.

**rm_map**
Removes a map.

**entry_val**
Identifies the user name to which the map will resolve.

**interp**
Identifies the interpreter type or operating system on which entry_val is a valid user name. Specifying the value for interp as default indicates that the default entry_val should be used on any operating system that does not have a separate interp entry.

**map_name**
Identifies the mapping to be viewed or modified.

**Authorization**

To view maps: user

To create, edit or delete maps: super.

**Examples**

1. The following example lists all mappings:
   ```
   widmap list_maps
   ```
2. The following example adds a default entry to the $chris map:
   ```
   widmap add_entry chris default chris
   ```
   ```
   chris  solaris2  chris
   chris  w32-ix86  chris_sanders
   chris  default  chris
   ```
3. The following example removes the solaris2 entry from the $chris map:
   ```
   widmap rm_entry chris solaris2
   ```
4. The following example returns the user name mapped in the $chris map to the default interpreter type:
   ```
   widmap resolve_entry chris default
   ```
   The result of this command is chris.

**Note:** If the same command were run for the HP-UX operating system, the result would be the same. Because there is not an entry for HP-UX, the default value is returned.
wifconfig

Queries or changes the Internet Protocol (IP) interfaces on a managed node.

Syntax

```
  wifconfig -h node_name
  wifconfig -h node_name -a device IP_address name notify_server
  wifconfig -h node_name -r device [IP_address] [name]
  wifconfig -h node_name -s device IP_address name notify_server
```

Description

The `wifconfig` command allows you to add, remove, or list the IP interfaces on a managed node.

Options

- `--a` Adds an IP interface to a UNIX managed node. You must specify a device, IP address, and name for the interface. You must also indicate whether the Tivoli server should be notified about the new IP interface. This option is supported only on UNIX managed nodes.
- `--h node_name` Specifies the managed node where the operations are performed.
- `--r` Removes an IP interface from a UNIX managed node. You must specify the device name. You can optionally specify the IP address and interface name. This option is supported only on UNIX managed nodes.
- `--s` Changes the settings of a current IP interface on the UNIX managed node. You must specify the device name and the new interface definition (IP address, interface name, and whether the Tivoli server is notified about the interface). You cannot change the device name with this option. This option is supported only on UNIX managed nodes.

```
device  Specifies the device used by the interface.
IP_address  Specifies the IP address of the interface.
name  Specifies the name of the interface.
notify_server  Indicates whether the Tivoli server in the local Tivoli region is notified of the addition or modification of the IP interface. This value must be TRUE or FALSE.
```

Authorization

To list: *user, admin, senior, super.*

To add or remove IP interfaces or to change properties of an IP interface: *admin, senior, super.*

Examples

1. The following example queries the IP address of the managed node bald:
wifconfig -h bald

<table>
<thead>
<tr>
<th>Device</th>
<th>Address</th>
<th>Name</th>
<th>Used by dispatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo0</td>
<td>127.0.0.1</td>
<td>localhost</td>
<td>unused</td>
</tr>
</tbody>
</table>

2. The following example adds the lel IP interface with the name bald2 with IP address of 146.84.49.3 to managed node bald and notifies the Tivoli server of the new interface:
   ```bash
   wifconfig -h bald -a lel 146.84.49.3 bald2 TRUE
   ```

3. The following example removes IP interface bald2 from managed node bald:
   ```bash
   wifconfig -h bald -r lel 146.84.49.3 bald2
   ```

See Also

netstat, `odadmin`
winsblk

Inserts a block of statements into a file. This command should be run from an endpoint.

Syntax

winsblk -s "search_string" -a "insertion_string" | @file_name [-o output_file] file_name
winsblk -s "search_string" -b "insertion_string" | @file_name [-o output_file] file_name

Description

The winsblk command inserts a block of statements into a file. This command enables you to add a block of statements delimited by unique strings, which you can later search for or remove using the wrplblk or wclrblk commands.

Options

-a "insertion_string" | @file_name
Inserts a block of statements after the line containing the search string. This option cannot be specified with the -b option.

If you specify insertion_string, you must surround the string with double quotation marks. If you specify @file_name, this command inserts the block of statements from the specified file.

-b "insertion_string" | @file_name
Inserts a block of statements before the line containing the search string. This option cannot be specified with the -a option.

If you specify insertion_string, you must surround the string with double quotation marks. If you specify @file_name, this command inserts the block of statements from the specified file.

-o output_file
Writes that output to the specified file. If this option is omitted, output is written to standard output.

-s "search_string"
Specifies a search string. If the search string is contained in a line, the block of statements is placed either before (using the -b option) or after (using the -a option) the line containing the search string. If this option is not specified, the block of statements is appended to the file.

file_name
Specifies the file to insert the block into.

Return Values

winsblk returns one of the following:

0 Indicates the command added the specified block of statements.
nonzero Indicates that the command did not add the specified block of statements.
Examples

To insert the statements in the c:\temp\blkstms.fil file after every occurrence of
the device= string in the c:\windows\system.ini file and redirect output to the
file, enter the following command:

winsblk -s "device=" -a @c:\temp\blkstms.fil \n-0 c:\temp\outputl.fil c:\windows\system.ini

See Also

wclrblk|winsline|wrplblk
**winsline**

Inserts a single line into a file. This command should be run from an endpoint.

**Syntax**

\[
\text{winsline} \ [-f] \ [-s \ "\text{search\_string}\"] \ [-a \ "\text{insertion\_string}\"] \ [-o \ \text{output\_file}] \ \text{file\_name}
\]

\[
\text{winsline} \ [-f] \ [-s \ "\text{search\_string}\"] \ [-b \ "\text{insertion\_string}\"] \ [-o \ \text{output\_file}] \ \text{file\_name}
\]

**Description**

The **winsline** command adds a line to a text file. You can insert the line before or after a line that contains the search string.

**Options**

- \( \text{–a \ "\text{insertion\_string}\"} \)
  
  Inserts the specified string or the lines contained in the specified file after the line that contains the search string. This option cannot be specified with the \( \text{–b} \) option. You must surround the string with double quotation marks.

- \( \text{–b \ "\text{insertion\_string}\"} \)
  
  Inserts the specified string or the lines contained in the specified file before the line that contains the search string. This option cannot be specified with the \( \text{–a} \) option. You must surround the string with double quotation marks.

- \( \text{–f} \)
  
  Processes only the first occurrence of the search string. If this option is omitted, the command processes each occurrence of the search string.

- \( \text{–o \ \text{output\_file}} \)
  
  Specifies the name of the file that output is written to. If this option is omitted, output is written to standard output.

- \( \text{–s \ "\text{search\_string}\"} \)
  
  Specifies the search string. If the search string is contained in a line, the insertion string is placed either before (using the \( \text{–b} \) option) or after (using the \( \text{–a} \) option) the line containing the search string. You must surround the string with double quotation marks.

- \( \text{file\_name} \)
  
  Specifies the name of the file to insert the line into.

**Return Values**

The **winsline** command returns one of the following:

- 0  
  
  Indicates that the command added the specified line.

- nonzero
  
  Indicates that the command did not add the specified line.

**Examples**

1. The following example inserts \texttt{lp01} after the first occurrence of the \texttt{device=} string in the \texttt{c:\windows\system.ini} file and redirect output to the \texttt{c:\temp\output.fil} file:

\[
\text{winsline} \ -f \ -s \ "\text{device=}" \ -a \ "\text{lp01}\" \ -o \ c:\temp\output.fil \ c:\windows\system.ini
\]
2. The following example inserts dev01 before every occurrence of the type= string in the c:\windows\system.ini file and redirect output to the c:\temp\output.fil file:

```
winsline -s "type=" -b "dev01" -o c:\temp\output.fil c:\windows\system.ini
```

See Also

wclrline, “winsblk” on page 230, wrplline
winstall

Installs a Tivoli Enterprise product.

Syntax

```
  winstall [–c source_dir] [–s server] [–i product] [–y] [install_variables] [–n] [managed_node...]
```

Description

The `winstall` command installs a Tivoli Enterprise product from the command line when invoked on the Tivoli region. Before installing any product, this command identifies the actions that are performed during the installation.

Options

- `–c source_dir`
  Specifies the complete path to the directory containing the installation image. For `source_dir`, you can specify either the fully qualified path (for example `/cdrom`) or the source host and the path in the format `source_host:path` (for example `oak:/cdrom`).

- `–i product`
  Specifies the product index file from which the product is installed. Index files have an .IND extension.

- `–n`
  Installs the product on all managed nodes that do not currently have the product installed. This option is ignored when `managed_node` is specified.

- `–s server`
  Specifies the managed node to use as the installation server. If not specified, the server is the Tivoli server.

- `–y`
  Installs the product without requesting confirmation.

- `install_variables`
  Specifies product-specific `keyword=value` pairs. For details, see "Installation Variables."

  Note: Type the names of the installation options exactly as specified in the product documentation. Installation options are case sensitive.

- `managed_node`
  Specifies the managed nodes on which to install this Tivoli Enterprise product. You can specify multiple managed nodes. If you do not specify a managed node, the product is installed on all managed nodes in this Tivoli region.

Installation Variables

You specify values for installation variables on the command line to control the installation. If a Tivoli Enterprise product has installation variables, you can view the index file for that product for its definitive list of installation variables. Use these variables to specify required and optional options as well as to override default installation information.

You can use installation variables to specify the directories where a Tivoli Enterprise product is to be installed. Specify these directories when you install Tivoli Management Framework; other Tivoli products use the same directories as
Tivoli Management Framework. If a directory already contains the files, the files are not reinstalled. You can, however, force any of these directories to be overwritten by entering an exclamation mark (!) as the value for the specified directory. For example, to reinstall the binaries, you would enter BIN=! instead of entering the entire path to the binaries directory. This override feature applies to all installation variables.

The following are the variables related to installation directories:

- **BIN=binaries_dir**
  Overrides the default installation path for the product binaries (/usr/local/Tivoli/bin).

- **LIB=libraries_dir**
  Overrides the default installation path for the product libraries (/usr/local/Tivoli/lib).

- **DB=client_database**
  Overrides the default installation path for the product client database (/var/spool/Tivoli).

- **MAN=manpage**
  Overrides the default installation path for the product man pages (/usr/local/Tivoli/man).

- **CAT=message_catalog**
  Overrides the default installation path for the product message catalogs (/usr/local/Tivoli/msg_cat).

The following is another useful variable:

- **@CreatePaths@=0 | 1**
  Indicates whether to create (1) or not to create (0) any specified directory if it does not already exist. By default, directories are not created. You will receive an error message if a specified directory does not exist.

**Authorization**

Requires `install_product` or `senior` in the Tivoli region

**Examples**

1. The following example installs Tivoli User Administration on all managed nodes in the Tivoli region, where the path to the installation image is /cdrom, and the product index file is ADMIN.

   ```
   wininstall -c /cdrom -i ADMIN
   ```

2. The following example installs Tivoli Software Distribution on managed nodes dan and barney, where the path to the installation image is /dev0/cdrom, and the product index file is COURIER:

   ```
   wininstall -c /dev0/cdrom -i COURIER dan barney
   ```

3. The following example reinstalled Tivoli User Administration on managed node petra, overwriting existing files in the binary directory, where the path to the installation image is /cdrom and the product index file is ADMIN:

   ```
   wininstall -c /cdrom -i ADMIN BIN=! petra
   ```

**See Also**

- `wclient`
- `wpatch`
- `wserver`
winstdir

Prints the path of the installation directory of the specified managed node.

**Syntax**

```
winstdir host_name
```

**Description**

The `winstdir` command prints the path of the installation directory for the specified managed node.

**Options**

```
host_name
```

Specifies the name of the host whose installation directory to list.

**Authorization**

user, admin, super, senior

**Examples**

The following example shows the installation directory for the managed node bald:

```
winstdir bald
```

```
/data/shadow/solaris2/as/bin
```

**See Also**

`wdate`, `wdiskspace`, `whostid`, `wifconfig`, `winterp`, `wmannode`, `wmemsize`, `wping`, `wtimezone`, `wuname`, `wxterm`
winstendpt

Installs behavior for an endpoint resource type.

Syntax

`winstendpt behavior [resource_type]`

Description

The `winstendpt` command installs behavior onto an Endpoint resource type. If a profile-based application defines an endpoint behavior, that behavior can be added to the inheritance of an Endpoint resource type such as a managed node. This command would typically be run from the initialization script of a profile-based application.

Options

- **behavior**
  Identifies the instance manager of the new endpoint behavior resource type.

- **resource_type**
  Identifies the endpoint resource type on which the new behavior is installed. If this option is omitted, the ManagedNode resource type is assumed.

Authorization

- super, senior

Examples

The following example installs the endpoint behavior of resource type aef onto the ManagedNode resource type:

```
aef_CO=`wlookup -r Classes aef`
managednode_CO=`wlookup -r Classes ManagedNode`
winstendpt $aef_CO $managednode_CO
```
winstlcf

Installs an endpoint on all operating systems except AS/400, and OS/2.

Syntax

```
```

Description

The `winstlcf` command installs and starts the endpoint service (lcfd) on one or more workstations. This command can be used to install an endpoint on all operating systems except AS/400 and OS/2 operating systems. For AS/400 operating systems, use the `w4inslcf.pl` command or Tivoli Software Installation Service. For OS/2 operating systems, use the provided InstallShield program or Tivoli Software Installation Service.

Notes:

- Before installing Linux endpoints, make sure that the the access method used (exec or shell) is enabled on the target. The exec access method is the default installation option; the shell access method is specified using the `–e` option.
- For UNIX operating systems, the login shell of the root user cannot be the C shell.
- To install Windows endpoints using the `winstlcf` command, you must first install one Windows endpoint in the domain or trust manually using the InstallShield image. This endpoint is used as a proxy to remotely install all additional Windows endpoints in the domain or trust. After you have installed the proxy endpoint, install all additional Windows endpoints in that domain or trust using the `winstlcf` command and `–N` option.

By default, the installation service starts after installation. You can install endpoints to multiple workstations by listing the machine names on the command line or using the `–f` option to specify a file that contains a list of machine names.

If you run the `winstlcf` command on a machine more than once, you have more than one instance of lcfd service running on that machine.

After you specify an installation password with the `winstlcf` command, that password becomes the default for all subsequent installations. To change the password, follow these steps:

1. Explicitly specify another password.
2. Attempt an installation on an unsupported operating system, which erases the global variable containing the password.
3. Specify the `–P` option.

Options

- `–a` Specifies that endpoints be installed asynchronously. Without this option, the command waits for the endpoint to log in to its gateway before installing the next endpoint.
For Linux and UNIX operating systems only, specifies the language locale for the target endpoint. If the –C option is not specified, the language is inherited from the lcfd environment.

Specifies the target directory in which to install the endpoint software. The default location is the /opt/Tivoli/lcf directory for UNIX operating systems and the c:\Program Files\Tivoli\lcf directory for Windows operating systems. When installing a Windows endpoint from a Linux or UNIX Tivoli server, forward slashes in path names are also supported.

For Windows operating systems, indicates that the lcfep program is not installed. When installing in a Windows Terminal Server environment, this is the default behavior and cannot be overridden. In other words, the lcfep program cannot be installed in this environment.

For UNIX operating systems only, specifies to use trusted host access instead of exec.

Specifies the name of a file that contains a list of machine where an endpoint needs to be installed. The file must contain one machine name per line. Each line can contain the machine name, the user ID, the password, the policy region, and the label that is used to install the endpoint. The following is the format of a line:

```
host_name user_ID password policy_region endpoint_label
```

where:

- **host_name**: The host name of the machine where the endpoint is to be installed.
- **user_ID**: The user ID of the system administrator performing the installation.
- **password**: The password associated with the user ID.
- **policy_region**: Optional. The name of the policy region where the endpoint is moved.

**Note:** If you specify endpoint_label and you do not want the endpoint moved to a policy region, you must specify policy_region as a null string (""").

- **endpoint_label**: Optional. The label of the endpoint.

For example, the following could be three lines in a file:

```
red root mstr_Key
orange chris dinomite "" orange-ep
yellow root mstr_Key NYC_PR yellow-ep
```

Specifies the Internet Protocol (IP) address or host name and, optionally, the port number of the gateway to which the endpoint logs in. Multiple
winstlcf

gateway entries must be separated by colons (:). You must specify the port number if it is other than 9494, the default. If the –g option is omitted, the endpoint broadcasts to all gateways.

Note: In a network address translation (NAT) environment, gateways must be specified as fully qualified domain names and not as IP addresses. Direct specification of gateway IP addresses fails in a NAT environment.

-i Turns off auto-start configuration for a Linux or UNIX endpoint after installation. By default, Windows endpoints always start automatically after installation.

-j Causes the command to use an encrypted secure shell (SSH) connection when connecting to the machine to install the endpoint.

Notes:
1. You cannot use SSH to perform installations on Windows targets.
2. When you use this option to install an endpoint from a Windows system, you must use a trusted host access method. Because trusted host access methods do not use passwords, you cannot use the –j option with winstlcf options that require a password.
3. To install an endpoint from a Windows system using the winstlcf command and –j option, you must launch a Cygwin command prompt and then run the winstlcf command from the version of Perl provided by Cygwin, as shown in the following example:
   /usr/bin/perl -S winstlcf -j options

For more information, see the chapter about SSH in the Tivoli Enterprise Installation Guide.

-l endpoint_port
   Specifies the port number for the endpoint. The default port number is 9495.

-L config_options
   Passes configuration options to the lcfd command for starting the endpoint. If you specify multiple options or have spaces in a single option, you must enclose the text in double quotation marks (”). See the lcfd command for a list of valid options.

-n endpoint_label
   Specifies an endpoint label provided by a user.

Note: If you omit the –n option, the endpoint label is generated automatically. If you do not specify the endpoint port number, the label is the host name of the endpoint. If you specify the endpoint port number (for example, using the –l option), the endpoint label is generated as follows:
   • On Windows operating systems, the label has the format host-port.
   • On Linux and UNIX operating systems, the label has the format host-instance, if the instance number is greater than 1. The value of instance matches the instance number used in the $LCFROOT/dat/instance directory.
-N endpoint
  Specifies an existing Windows endpoint in the domain or trust to be used as a proxy to remotely install all other Windows endpoints.

Note: To install Windows endpoints using the winstlcf command, you must first install one Windows endpoint in the domain or trust manually using the InstallShield image. This endpoint is used as a proxy to remotely install all additional Windows endpoints in the domain or trust. After you have installed the proxy endpoint, install all additional Windows endpoints in that domain or trust using the winstlcf command and –N option. When you use this option, all endpoints to be installed are assumed to be Windows clients. Installing the Tivoli Remote Execution Service is not necessary.

-P Prompts for a password for each machine. This option is useful only when installing on remote hosts with different passwords. If each machine has the same password or if you do not use this option, the command prompts for a global password to use for each machine.

-r policy_region
  Specifies a policy region to install the endpoint to.

-R Requires the Windows endpoint to restart after installation without prompting the user. This option is only needed if the Tivoli Authentication Package, TivoliAP.dll, was not previously installed on the endpoint or an older version of the Tivoli Authentication Package is being replaced.

-s dir_name
  Specifies the source directory containing the endpoint installation image.

-S share_name
  Specifies a destination share name (default = C$).

-T account
  Specifies the Tivoli remote access account for the Windows endpoint.

-v Lists verbose installation information and error messages.

-x [TCP/IP | IPX]
  For Internetwork Packet Exchange (IPX) endpoints only. Specifies the protocol used by the endpoint. If you do not specify this option, the endpoint uses TCP/IP. Supported protocols are TCP/IP and IPX. To specify both TCP/IP and IPX, specify the option as -x=TCP/IP,IPX. You cannot turn off the TCP/IP protocol for a gateway.

-Y Installs the endpoint without confirmation. By default, this command identifies the actions that must be taken to perform the installation and requests confirmation before continuing. Using this option, the command identifies the actions and performs the installation without requesting confirmation.

host [user_account password]
  Specifies the name of the machine on which the endpoint is installed. If you specify only the host name, the root or Administrator account is used. You are prompted for the password. You can specify a different user account and password by enclosing the three entries in single quotation marks. For example, you might enter the following:

winstlcf 'vernon DOMAIN-NT\chris dln0mite'
winstlcf

If the Windows domain and the local computer use the same user_account name (such as Administrator), you must specify the fully qualified name for the account, as in the preceding example. Quotation marks are necessary when specifying fully qualified user accounts.

Authorization

No Tivoli authorization role is required except when the –N option is specified. To use the winstlcf command and –N option, you must have the super, senior, or admin role.

Examples

1. The following example installs the endpoint software on a UNIX workstation vernon, sets the locale to French, and starts the endpoint daemon (lcfd). The winstlcf command uses the root account and prompts for the root password on vernon. The installation image is placed in the default directory. The endpoint starts with the default configuration.
   winstlcf -C fr vernon

2. The following example installs the endpoint software on a Windows workstation olympus and starts the endpoint service. The winstlcf command uses the Administrator account and prompts for the Administrator password on olympus. The installation image is taken from a Windows proxy fuji (a previously installed endpoint in the Windows domain). The software is installed in the default directory on olympus. The endpoint starts with the default configuration.
   winstlcf -N fuji olympus

3. The following example installs the endpoint on a Windows workstation in a directory other than the default directory. In this example, the endpoint is installed on workstation bonnell on drive D with the share name steve. For instances where the share name of the destination drive is not the default name (D:\ = D$), use the –d to specify the directory (D:\tivoli\lcf), and use the –S to specify the share name (steve).
   winstlcf -N pctmp107 -d D:\tivoli\lcf -S steve bonnell

4. The following example installs the endpoint software on workstation myoung. The endpoint performs its initial login through IP address 123.45.1.12.
   winstlcf -g 123.45.1.12 myoung

5. The following example installs the endpoint on workstation bbunny and passes configuration options to the lcfd command to use when it starts the endpoint. In the example, -g cedar+1616 specifies the gateway and port that the endpoint contacts for initial login, and -Dlcs.machine_name=bbunny-ep assigns a specific name to the endpoint.
   winstlcf -L "-g cedar+1616 \ -Dlcs.machine_name=bbunny-ep" bbunny

6. The following example installs machines cedar and mahogany as endpoints. The installation process prompts for a global root password, but does not prompt for confirmation before installing.
   winstlcf -P -Y cedar mahogany

7. The following example installs multiple endpoints from the endpt.txt file. The installation process does not prompt for password or installation confirmation. The software is installed in the /usr/lcf directory.
   winstlcf -f endpt.txt -Y -d /usr/lcf
8. The following example installs the Windows endpoint antonella on the computer system agodino using IPX to connect to the NetWare gateway lux using the endpoint vernon as a proxy.
   `winstlcf -x IPX -N vernon -g LUX+7787 -n antonella agodino`

9. The following example installs an endpoint on machine oak using an encrypted SSH connection from a Windows managed node.
   `/usr/bin/perl -S winstlcf -j oak`

**Note:** To install an endpoint from a Windows managed node using the `winstlcf` command and `-j` option, you must launch a Cygwin command prompt and then run the `winstlcf` command from the version of Perl provided by Cygwin.

**See Also**

`lcfd.sh wdelep`
winterp

Prints the interpreter type of the specified managed node.

Syntax

```
  winterp host_name
```

Description

The `winterp` command prints the interpreter type for the specified managed node.

Options

```
host_name
```

Specifies the name of the host for which to list the interpreter type.

Authorization

```
user, admin, senior, super
```

Examples

The following example shows the interpreter type of the managed node bald:

```
  winterp bald

  solaris2
```

See Also

```
wdate, wdiskspace, whostid, wifconfig, wmannode, wmemsize, wping, wtimezone, wuname, wxterm
```
wlcftap

Sets the properties of the Tivoli Authentication Package on a Windows endpoint.

Syntax

```bash
wlcftap [-a | -d] [-B] [-P] [-r [domain_name\user_name | user_name]] [-k]
```

Description

The `wlcftap` command sets the properties of the Tivoli Authentication Package, TivoliAP.dll, on the local endpoint. This file enables Tivoli Management Framework to access remote file systems in the context of a user. It enables a Windows operating system to run `setuid` methods, that is, to run a method in the context of a user associated with the method. Refer to `Tivoli Management Framework Planning for Deployment Guide` for more information about accessing remote file systems.

The Tivoli remote access account specifies a user account. Tivoli uses this account to access remote file systems.

Using the `wlcftap` command with no options prints version information from the currently running TivoliAP.dll.

When activating the Tivoli Authentication Package for the first time, which is usually immediately following Tivoli server installation, you must restart the machine for changes to take effect.

**Note:** In a Windows 2000 domain environment that uses Active Directory, Tivoli Authentication Package uses the Tivoli remote access account to connect to Active Directory rather than using an anonymous login. To use this feature, you must set the Tivoli remote access account to a valid domain-qualified user ID and password.

Options

- `-a` Sets the Tivoli Authentication Package internal key and registers the TivoliAP.dll with the local security authority (LSA). The new internal key becomes effective immediately. The TivoliAP.dll file is loaded by the LSA when the machine is restarted.

- `-B` Authenticates domain users using a domain controller other than the primary domain controller. To authenticate users using the primary domain controller, use the `-P` option.

- `-d` Removes the Tivoli Authentication Package internal key and unregisters the TivoliAP.dll with the LSA. The TivoliAP.dll file is released by the LSA (so that it can be deleted if Tivoli Management Framework is uninstalled) when the machine is restarted.

- `-k` Indicates that the command should read the password for `user_name` from standard input. If you do not specify this option, the command prompts you for the password.

- `-P` Authenticates domain users by using the primary domain controller. This is the default setting. To authenticate users by using other domain controllers, such as backup domain controllers, use the `-B` option.

- `-r [domain_name\user_name | user_name]` Sets the Tivoli remote access account to the specified user name. Tivoli
**wlcftap**

accesses remote file systems using this user account. This user name can be prefixed with the domain name, separated by either a forward slash (/) or a backward slash (\). If the domain is specified, it must be the domain in which the machine running the Tivoli Authentication Package belongs or a domain trusted by that domain. If no domain is specified, Windows looks for the given user in the local domain or trusted domains. Running the **wlcftap** command with the `-r ""` option disables Tivoli access to remote file systems. To see changes take effect immediately, restart the object dispatcher.

**Authorization**

Member of the Administrators group, Tivoli **admin** authorization is required to run **wlcftap** with no options, and Windows operating systems permission to edit the registry.

**Examples**

The following example sets the Tivoli remote access account to the userTME user account. The **wlcftap** command reads the account password from the passwd.txt file.

```
wlcftap -r userTME -k < passwd.txt
```
Links an object into a collection.

**Syntax**

```
wln [-I] label... collection
```

**Description**

The `wln` command links an object with the specified label into the specified collection. The `label` and `collection` options can both be full or partial label paths.

If you use this command to link an endpoint, run the `wep sync_gateways` command to synchronize the endpoint data stored by the endpoint manager, gateways, and endpoints within the Tivoli region.

**Options**

- `-I` Ignores all failed suboperations, allowing the command to continue. This option is useful only when multiple labels are passed to the command. The `-I` option allows a link operation to fail for individual objects, but the command continues to the next object to be linked. Without this option, if a link operation fails for an individual object, the command unlinks any objects already linked, and then the command terminates with error. The default is for the command to fail when the suboperation fails.

- `collection` Specifies the label of the collection into which the object is to be linked. This option also can be a full label path (starting at the root `/`) collection), a partial label path (relative to the current working collection), or a simple name (to be found in the current working collection). The linked object becomes a member of the selected collection.

- `label` Specifies the label of the object. This option can be a full label path (starting at the root `/`) collection), a partial label path (relative to the current working collection), or a simple name (to be found in the current working collection).

**Authorization**

admin, senior, super

**Examples**

The following example creates a new administrator using the `wcrta admin` command and then uses the `wln` command to link the Tivoli server and scheduler objects from the default root desktop into the newly created administrator desktop collection:

```
wcrta admin -a jack -r global,backup:admin:user \
 -r @ceridwen-region,admin:senior:user \
 -r @Administrators,admin:user -r @Scheduler,admin:user \
 -n "Tivoli Administration" -n "Tivoli Authorization" \
 -n "Tivoli Diagnostics" -n "Tivoli Scheduler" -u jack -g staff \
 "Jack Frost"
wln /Administrators/Root_ceridwen-region/ceridwen-region \
 /Administrators/"Jack Frost"
wln /Administrators/Root_ceridwen-region/Scheduler \
 /Administrators/"Jack Frost"
```

Chapter 3. Commands alphabetically  247
See Also

wep  wmv
wlocalhost

Sets the name of the local host in the Windows registry.

Syntax

    wlocalhost [host_name]

Description

The wlocalhost command sets the local host name in the Windows registry. If you do not specify a host name, the command returns the host name currently stored in the registry.

Options

    host_name

    Specifies the name of the local host.

Authorization

Member of the Administrators group
wlocktmr

Places the current Tivoli region in maintenance mode.

Syntax

wlocktmr –p

wlocktmr –e command_file

Description

The wlocktmr command places the current Tivoli region in maintenance mode. In this mode, you can perform various maintenance and diagnostic tasks. Note that issuing this command immediately terminates all active Tivoli processes.

You can invoke wlocktmr in two ways. The first way is to place the Tivoli region in maintenance mode indefinitely. To do this, use the –p option. This option puts the Tivoli region in maintenance mode and pauses until you manually stop the process. You can perform maintenance and diagnostic operations in another window.

The second way to invoke wlocktmr is to run a single maintenance command or to run a batch or script file. To do this, use the –e option. This option puts the Tivoli region in maintenance mode, runs the specified command or file, and exits maintenance mode.

Use the vi command (or any text editor) to create the list of commands. Use the chmod command to make sure that the commands are executable. Then run the wlocktmr command with the –e option and the command file name.

Options

–e command_file

Places the Tivoli region in maintenance mode, runs the command given by command_file, and exits maintenance mode. The command_file can be the name of a batch or script file.

–p

Places the Tivoli region in maintenance mode and pauses until you manually stop the process.

Authorization

super
wlocpath

Returns the path for the localized file or directory.

**Syntax**

```bash
wlocpath path [-d default_path] [-l locale] [-o output]
```

**Description**

The `wlocpath` command searches for the localized file or directory and then prints the path to standard output. If the path is not found, the command returns nothing.

**Note:** This is the command line interface (CLI) for the `tis_get_loc_path()` function.

The `wlocpath` command uses the LANG environment variable to find the message catalog appropriate for the current localized environment (locale). For example, if LANG is equal to fr_FR and NLSPATH is equal to

```
/tivoli/msg_cat/%L/%N.cat;/tivoli/msg_cat/%L%N.cat;/tivoli/msg_cat/C/%N.cat
```

the `wlocpath` command tries the following path names to find the message catalog:

```
/tivoli/msg_cat/fr_FR/catalog_name.cat
/tivoli/msg_cat/fr/catalog_name.cat
/tivoli/msg_cat/C/catalog_name.cat
```

After the message is retrieved and bound, the resulting string is written to standard output.

**Options**

- `-d default_path`
  Identifies the default path. If the command to obtain the valid path from the path option fails, this option is used.

- `-l locale`
  Identifies the locale name. If this option is not specified, the value of the current LANG environment variable is used.

- `-o output`
  Identifies the name of the output file instead of writing to the standard output.

- `path`
  Inputs the path string. If this string contains the %L variable, it is substituted with the locale name.

**Authorization**

This command does not require any Tivoli authorization roles.

**Examples**

The following example looks for the query.txt file in the ./msg_cat/%L directory:

```bash
wlocpath ./msg_cat/%L/query.txt -d/tmp -l -fr
```

The result is what the LANG environment variable is for this file. If this variable is not found, the -d option looks in the /tmp directory. If the variable is still not found, the -l option looks in the /fr directory.
wlookup

Searches for a resource instance in the Tivoli name registry.

Syntax

```
wlookup [-l] -R
wlookup -a [-L | -o] -r resource_type
wlookup -r resource_type resource_name
wlookup -r resource_type -n instance_name {resource_name | -a}
```

Description

The `wlookup` command searches the Tivoli name registry for object information about a resource. If no type is specified, the default resource type is distinguished. If neither the `-L` nor `-o` option is specified, both the object identifier and label of the specified resource are returned. If the `-l` and `-R` option are specified, the time stamps for the modified resource types are listed.

Options

```
-a     Lists all instances of the specified resource type in the name registry.
-l     Lists the dates that the resource types were modified.
-L     Lists the label of the specified resource without its object identifier.
-n instance_name  Looks up a nested resource under the specified instance.
-o     Lists the object identifier of the specified resource without its label.
-r resource_type  Specifies the resource type to be retrieved. If omitted, the default resource type is distinguished.
-R     Displays a list of all resource types that are registered.
```

Authorization

`user, admin, senior, super`

Examples

1. The following example looks up all distinguished resources in the Tivoli name registry:
   ```
wlookup -a
   ```
2. The following example displays a list of all registered resource types:
   ```
wlookup -R
   ```
3. The following example displays all instances of the policy region resource type:
   ```
wlookup -r PolicyRegion -a
   ```
4. The following example displays the object information for a particular instance (MyTask) of a resource type (TaskLibrary):
5. The following example displays the TaskExecute resources on managed node vail:

```
wlookup -r ManagedNode -n vail TaskExecute
```

See Also

[wrregister]
wls

Lists the member objects of a collection.

Syntax

\texttt{wls [-d] [-l] [-o] \[\textit{path}\]}

Description

The \texttt{wls} command lists the members of the selected collection.

Options

\texttt{\texttt{-d}} Lists information about the collection itself, but not about the collection members.
\texttt{\texttt{-l}} Lists the object identifier and the label of each member.
\texttt{\texttt{-o}} Lists the object identifier of each member.
\texttt{\textit{path}} Specifies the path to the collection object whose members are to be listed.
Valid formats for the path are as follows:
\begin{itemize}
  \item @\textit{resource_type}:\textit{collection_name}
  \item /\textit{region}/\textit{policy_region_name}/\textit{collection_name}
\end{itemize}

The default collection is the current working collection.

If the \texttt{-l} and \texttt{-o} options are omitted, this command lists only the member labels.

Authorization

You must have at least \texttt{user} in a group that the collection is a member of.

Examples

1. The following example lists the members of the current working collection:
\begin{verbatim}
wls
\end{verbatim}

2. The following example lists the object identifiers and the labels of the members of the Administrators collection. This command is helpful in identifying administrators (or other objects) removed from the desktop but not deleted from the Tivoli database.
\begin{verbatim}
wls -l /Administrators
\end{verbatim}

3. The following example lists the objects on the desktop of administrator Jorge:
\begin{verbatim}
wls @Administrator:jorge
\end{verbatim}

See Also

\texttt{wcd wpwd}
wlsconn

Lists the current Tivoli region connections or information about a single connection, and completes the exchange of connection information when the connection process could not.

Syntax

wlsconn [region_name]
wlsconn –u region_name

Description

The wlsconn command lists current Tivoli region connection information including region number, port, connection mode, and resources and when they were last exchanged. If the –u option is specified, the command completes the exchange of connection information between the local region and the connected remote region. If no options are specified, the command lists all current connections with the local region.

When exchanging resources between connected regions, it is useful to compare the output from the wlookup –l –R command to the output from the wlsconn command to determine which resources to update.

If the –u option is specified, wlsconn completes the exchange of connection information if the connection process could not. If you use secure procedures to create a one-way connection in which the first side is the managing server and the second side is the managed server, you must use this command with the –u option to exchange connection information.

Options

–u region

Updates the Tivoli region name, remote server name, and other cached interregion information with the current information from the specified remote Tivoli region.

region_name

Specifies the name of the remote Tivoli region.

Authorization

super if the –u option is specified; otherwise, user.

Examples

1. The following example lists all current connections to the local Tivoli region:

   wlsconn

   MODE  NAME           SERVER    REGION
   <->   morie-Region   morie     3333333333
   ---->  anon-sul-Region anon-sul 5555555555
   <---- ceridwen-Region ceridwen 2222222222

2. The following example lists the connection information for the connection between the local Tivoli region and the remote morie-Region Tivoli region:
wlsconn

wlsconn morie-Region

Name: morie-Region
Server:morie
Region:3333333333
Mode:two_way
Port:94

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Last Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMF_Notice</td>
<td>Fri Jan 09 11:33:10 1998</td>
</tr>
<tr>
<td>Administrator</td>
<td>Fri Jan 02 13:13:15 1998</td>
</tr>
<tr>
<td>PolicyRegion</td>
<td>Tue Jan 13 10:00:38 1998</td>
</tr>
<tr>
<td>TaskLibrary</td>
<td>Tue Nov 04 10:02:34 1997</td>
</tr>
<tr>
<td>Job</td>
<td>Wed Dec 31 19:00:00 1969</td>
</tr>
<tr>
<td>QueryLibrary</td>
<td>Wed Dec 31 19:00:00 1969</td>
</tr>
<tr>
<td>Query</td>
<td>Wed Dec 31 10:00:00 1969</td>
</tr>
<tr>
<td>ProfileManager</td>
<td>Wed Nov 05 17:49:38 1997</td>
</tr>
<tr>
<td>ManagedNode</td>
<td>Tue Jul 07 19:24:34 1998</td>
</tr>
<tr>
<td>Repeater</td>
<td>Thu Sep 04 10:04:32 1997</td>
</tr>
<tr>
<td>CheckDB</td>
<td>Thu Sep 04 20:04:23 1997</td>
</tr>
<tr>
<td>RemoveNode</td>
<td>Thu Sep 04 20:04:23 1997</td>
</tr>
<tr>
<td>HTTPRealm</td>
<td>Tue Nov 04 20:03:22 1997</td>
</tr>
<tr>
<td>HTTPRealmMakerGroup</td>
<td>Thu Sep 04 20:07:22 1997</td>
</tr>
<tr>
<td>HTTPRealmMaker</td>
<td>Thu Sep 04 20:07:23 1997</td>
</tr>
<tr>
<td>DependencyMgr</td>
<td>Thu Sep 04 20:07:23 1997</td>
</tr>
<tr>
<td>Gateway</td>
<td>Wed Dec 31 19:00:00 1969</td>
</tr>
<tr>
<td>Endpoint</td>
<td>Wed Dec 31 19:00:00 1969</td>
</tr>
</tbody>
</table>

3. The following example exchanges connection information between the local Tivoli region and the remote morie-Region Tivoli region

wlsconn -u morie-Region

See Also

wconnect wdisconn wlookup wupdate
wlsendpts

Lists all the endpoints subscribed to a profile manager.

Syntax

wlsendpts [-l] profile_manager

Description

The wlsendpts command lists all the endpoints that are directly or indirectly subscribed to the specified profile manager. The endpoints listed can be on the first subscription level (that is, profile manager to subscriber) or on any subscription level below that (that is, profile manager to profile manager or profile manager to endpoint).

Options

-[-l] Prints the object ID and label of the endpoint. Without this option, only the label is printed.

profile_manager

Specifies the name of the profile manager. If the name of the profile manager contain spaces, enclose the entire name in quotation marks. Valid formats for the name of the profile manager are as follows:

- @profile_manager_name
- @ProfileManager:profile_manager_name
- /Regions/policy_region_name/profile_manager_name

Authorization

user

Examples

The following example lists the endpoints subscribed to the Admin_PM profile manager:

wlsendpts @ProfileManager:Admin_PM

pepper (ManagedNode)
gumby (ManagedNode)
wlsinst

Lists the products and patches installed in a Tivoli region.

Syntax

wlsinst [-a | -l | -p | -P | -s name | -V] [-i] [-v] [-h]

Description

The wlsinst command lists the installed products, the installed patches, or both products and patches in the Tivoli region. Adding the -i, -v, or -h options includes host information with the product or patch list. The -s option searches for the specified product or patch.

Options

- Lists all products and patches installed in the Tivoli region.
  - Lists the host name and interpreter type of the machines where the products or patches are installed.
  - Sorts output by interpreter type. This option must be used with either the -v or -h option.
  - Lists all the patches installed for those products that have been patched. If none of the products have been patched, no output is given.
  - Lists all products installed in the Tivoli region.
  - Lists all patches installed in the Tivoli region.
- Lists the specified product or patch. When the name for the product or patch contain spaces, enclose the product or patch name in quotation marks.
- Lists all host names, interpreter types, and the directories in which the various components (for example: binaries, libraries, or man pages) of each product or patch were installed.
- Prints out the path to each component, replacing any space at the end of the path name with a slash.

Authorization

super, senior, admin, or user

Examples

1. The following example lists all products and patches installed in the Tivoli region:

   wlsinst -a

   *---------------------------------------------------*
   | Product List                                    |
   *---------------------------------------------------*
   Tivoli Management Platform
   Tivoli/Admin 2.5
   *---------------------------------------------------*
   | Patch List                                      |

   Patch List
2. The following example lists only the products installed in the Tivoli region:
   
   ```
wlsinst -p
   ```

   *---------------------------------------------------*
<table>
<thead>
<tr>
<th>Product List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tivoli Management Platform</td>
</tr>
<tr>
<td>Tivoli/Admin 2.5</td>
</tr>
</tbody>
</table>
   *---------------------------------------------------*

3. The following example lists the products installed, the host name and interpreter type on which each product was installed, and the directories in which each product was installed:
   
   ```
wlsinst -p -v
   ```

   *---------------------------------------------------*
<table>
<thead>
<tr>
<th>Product List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tivoli Management Platform</td>
</tr>
<tr>
<td>ida</td>
</tr>
<tr>
<td>solaris2</td>
</tr>
<tr>
<td>ALIDB /var/spool/Tivoli ida.db</td>
</tr>
<tr>
<td>APPD /usr/lib/X11/app-defaults</td>
</tr>
<tr>
<td>BIN /usr/local/Tivoli/bin solaris2</td>
</tr>
<tr>
<td>BIN /usr/local/Tivoli/bin solaris2</td>
</tr>
<tr>
<td>BUN /usr/local/Tivoli/bin client_bundle</td>
</tr>
<tr>
<td>CAT /usr/local/Tivoli/msg_cat</td>
</tr>
<tr>
<td>CONTRIB /usr/local/Tivoli/bin solaris2/contrib</td>
</tr>
<tr>
<td>GBIN /usr/local/Tivoli/bin generic_unix</td>
</tr>
<tr>
<td>LIB /usr/local/Tivoli/lib solaris2</td>
</tr>
<tr>
<td>tornado</td>
</tr>
<tr>
<td>solaris2</td>
</tr>
<tr>
<td>APPD /usr/lib/X11/app-defaults</td>
</tr>
<tr>
<td>BIN /usr/local/Tivoli/bin solaris2</td>
</tr>
<tr>
<td>CAT /usr/local/Tivoli/msg_cat</td>
</tr>
<tr>
<td>CONTRIB /usr/local/Tivoli/bin solaris2/contrib</td>
</tr>
<tr>
<td>DB /var/spool/Tivoli tornado.db</td>
</tr>
<tr>
<td>GBIN /usr/local/Tivoli/bin generic_unix</td>
</tr>
<tr>
<td>LIB /usr/local/Tivoli/lib solaris2</td>
</tr>
</tbody>
</table>
   ```
wlsnotif

Lists notices on the bulletin board of an administrator.

Syntax

wlsnotif

wlsnotif –g

wlsnotif –l [–n notice_group]

Description

The wlsnotif command lists the notices on the bulletin board of an administrator. The –g option lists all the notice groups. The –l option lists only the headers of notices instead of the entire message. The –n option lists notices from the specified notice group. If –n is omitted, the notices from all notice groups are listed. If no options are provided, all notices from all notice groups are listed. This command restricts the notice groups to the subset of notice groups that the administrator subscribes to. In all cases, the output from this command is written to standard output.

Options

–g  Lists the valid notice groups, one per line. When this option is used, it overrides the other options.

–l  Lists the headers of the notices on the administrator bulletin board. If this option is omitted, this command lists the actual notices. Output is truncated to 40 characters.

–n notice_group  Specifies the notice group from which notices are to be listed or the specific notices to list. If this option is omitted, this command lists notices from all notice groups.

Authorization

user, admin, senior, super

Examples

1. The following example lists all notices from all notice groups. This output could be extremely long, so it should be redirected to a file.

   #wlsnotif

   Date: Mon Nov 21 10:29:12 1994
   Notice-Group-Name: Tivoli Administration
   Priority: Notice
   Sent-By-Administrator: root@bald

   A new IP interface was added on bald by root@bald.
   device: le1
   address: 146.84.49.3
   name: bald2

2. The following example lists all the notification groups to which the administrator is subscribed:
wlsnotif -g

Tivoli Administration
Tivoli Authorization
Tivoli Diagnostics
Tivoli Scheduler

3. The following example lists the header of all messages for the administrator:

wlsnotif -l

11/21/01 10:29:12 CST A new IP interface was added on bald by
11/21/01 10:30:25 CST An IP interface was deleted from bald by
11/21/01 10:34:32 CST Deleted Objects
11/21/01 10:37:08 CST A new task, date_task, was created by ro
11/21/01 10:39:35 CST The task, date_task, of the my_tasks tas
11/21/01 10:49:33 CST The task, date_task2, of the my_tasks ta
11/21/01 10:50:45 CST The task, date_task2, of the my_tasks ta
11/21/01 10:55:52 CST A new task, find_cores, was created by r

See Also

wexpnotif wsendnotif wtailnotif
**wlspol**

Lists available policy default and validation objects for a Tivoli managed resource type.

**Syntax**

```
wlspol [-d] resource
wlspol -v resource
```

**Description**

The `wlspol` command lists the names of the policy default objects and policy validation objects for the specified managed resource type.

**Options**

- `-d` Lists the labels of the policy default objects for the specified managed resource type. Policy default objects generate default attribute values for resources created in a policy region. This action is the default unless the `-v` option is specified.

- `-v` Lists the labels of the policy validation objects for the specified managed resource type. Policy validation objects validate attribute values for managed resources.

`resource`

Specifies the managed resource type whose policy default objects or policy validation objects are to be listed.

**Authorization**

`senior`, `super`

**Examples**

The following example lists all the policy validation objects for the ProfileManager managed resource type:

```
wlsop -v ProfileManager
```

**See Also**

`wchkpol`, `wcrtpol`, `wcrtp`, `wdelpol`, `wdelpr`, `wgetdfpol`, `wgetpol`, `wlspol`, `wputpolm`
wlspolm

Lists policy methods for a Tivoli managed resource type or lists attribute names for a profile.

Syntax

wlspolm [-d | -v] class

wlspolm [-d | -v] profile

Description

The `wlspolm` command with the `class` option lists the names of the policy methods assigned to the specified managed resource type, which the `wlspolm` command with the `profile` option lists the attribute names (properties that can have policies associated with them) for the specified profile. The final option can be any managed resource type that supports policy methods (for example, a host, Network Information Services [NIS] domain, or file package) or a profile.

The names listed by this command can be used as input for the `wgetpolm` and `wputpolm` commands.

The `-d` option lists default policies (default), and the `-v` option lists validation policies. To set default and validation policies for specific Tivoli applications, see the documentation for the specific Tivoli application.

Options

- `-d` Lists the policy default methods. This action is the default unless the `-v` option is specified.
- `-v` Lists the policy validation methods.
- `class` Specifies the managed resource type whose policy methods are to be listed.
- `profile` Specifies the profile whose attribute names are to be listed.

Authorization

`senior` or `super`

Examples

1. For nonprofile usage, the following example lists all the policy methods on a ProfileManager policy validation object:

   `wlspolm -v ProfileManager`

   `pm_val_remove_subscribers`
   `pm_val_remove_subscription`
   `pm_val_subscribers`
   `pm_val_subscription`

2. For profile usage, the following example lists the properties that can have policies associated with them for the phone list profile Engineering:

   `wlspolm -d @PhoneListProfile:Engineering`

   `name`
   `phone`
   `address`
   `city`
wlspolm

state
country
postal
owner
type
comment

**See Also**

wchkpol, wcrtpol, wcrtpr, wdelpol, wdelpr, wgetdfpol, wlspol, wputpolm
wlssub

Lists the profile managers to which an object subscribes.

Syntax

```
wlssub -l [-o] name
```

Description

The `wlssub` command lists the profile managers to which an object subscribes. These objects can be a managed node, Network Information Services (NIS) domain, endpoint, profile manager or other object.

Options

- `-l` Specifies a long listing.
- `-o` Lists the object identifier for each profile manager to which an object subscribes. This object can include a managed node, an NIS domain, an endpoint, or a profile manager.

`name` Specifies the name of the object whose subscriptions are to be listed. Valid formats for this option are as follows:

- `@domain_name`
- `@NisDomain:domain_name`
- `/Regions/policy_region_name/domain_name`

Authorization

`user, admin, senior, super`

Examples

The following example lists all the profile managers to which managed node cook subscribes:

```
wlssub @ManagedNode:cook
```

See Also

`wcrtpf`, `wcrtpfmg`, `wdistrib`, `wgetprf`, `wgetsub`, “wpopulate” on page 291`, `wsupd`, `wvalidate`
wlstlib

Lists the contents of a task library.

**Syntax**

`wlstlib library_name`

**Description**

The `wlstlib` command lists the jobs and tasks contained in the specified task library.

**Options**

`library_name`

Specifies the name of the task library.

**Authorization**

`user, senior, super`

**Examples**

The following example lists the contents of the queue_lib task library:

```
wlstlib queue_lib
```

 Clean Queue  (task)
 Clean Queue  (job)

**See Also**

`wcrtlib`
**wmailhost**

Specifies the mail server used by Tivoli Management Framework on Windows operating systems.

**Syntax**

```
wmailhost [?] [host_name]
```

**Description**

The `wmailhost` command provides a Simple Mail Transfer Protocol (SMTP) connection for Windows operating systems in a Tivoli environment. Issue this command on any Windows Tivoli server or managed node. The Tivoli tools that generate e-mail send it to the specified mail server. Without the `host_name` option, the command returns the currently specified mail server.

The mail server must be an SMTP server or one connected with an SMTP gateway. For example, if the network mail server has Microsoft Exchange or Lotus Notes installed on a Windows 2000 computer, you must install the SMTP gateway software on the computer. For more information about configuring for SMTP e-mail, see *Tivoli Enterprise Installation Guide*.

**Options**

- `?` Prints the usage statement.
- `host_name` Specifies the host name of the mail server.

**Examples**

The following example sets the mail server to host loki:

```
wmailhost loki
```
wmannode

Returns system information about a managed node.

Syntax

wmannode node_name

Description

The wmannode command returns system information about a managed node in the Tivoli region. The managed node must be available when you issue this command.

Options

node_name

Specifies the name of a managed node.

Authorization

user, admin, senior, super

Examples

The following example lists the system information for managed node yankee:

wmannode yankee

System Name : yankee
Interpreter : solaris2
Install Directory : /usr/local/Tivoli
Host ID : 945bd30
System Architecture : sun4m
Memory Size (MB) : 48
System Timezone : 360
OS Name : SunOS
OS Release : 5.3
OS Version : Generic_101318-21

See Also

wdiskspace whostid wident winstdir winterp wmemsize wtimezone
wmcast

Configures repeaters for MDist 2 multicast distributions.

Syntax

```
wmcast -s [default | repeater_name] [keyword=value...]
wmcast -p {all | repeater_name}
```

Description

The `wmcast` command sets repeater properties for MDist 2 multicast distributions. The defaults provided are designed for use in most LAN environments. However, if you are using multicast over both fast links and slow links, configure multicast repeaters for the slowest link.

Options

- `–p` Sends a multicast ping to each endpoint connected to the specified gateway repeater.
- `–s` Configures a repeater using one or more of the following keywords and values. If a value is not specified, the existing options for the specified repeater are displayed.
- `all` Specifies all repeaters including the default setting.
- `default` Specifies the default setting for a new repeater.
- `repeater_name` Specifies the label or object ID of the repeater.
- `keyword=value` The keywords and values are as follows:

  **backofftm=time_in_milliseconds**
  Specifies the back off time in milliseconds. When receivers acknowledge receipt of a multicast advertisement, the receiver waits for a random time interval between 0 milliseconds and the number of milliseconds specified by this keyword before responding to the sender. This reduces congestion. As you add more receivers, this number might need to be increased to improve performance. The default is 100.

  **blocksize=size_in_bytes**
  Specifies the size of the block, in bytes, that the sender uses when writing data to the network. The size specified must be less than 65535 bytes. The default is 1460 bytes, which is the Maximum Transmission Unit (MTU) for Ethernet transmissions.

  **connrty=retry_count**
  Specifies the number of times that a multicast sender will rebroadcast the connection message to the receivers. The default is 5.

  **connwtout=milliseconds**
  Specifies the time, in milliseconds, that a multicast sender waits for receivers to accept a connection. The default is 2000.
dtrtry = retry_count
Specifies the number of times that a multicast sender will resend dropped packets to receivers. The default is 10.

dtwtout = time_in_milliseconds
Specifies the time, in milliseconds, that a receiver will wait before timing out if the data transmission is interrupted. The default is 3000.

ifrcvaddr = address...
Specifies a list of IP addresses that the receivers use when listening for multicast packets. Separate multiple addresses with semicolons (;) and enclosed in double quotation marks ("). If an address is not specified, the default is 0.0.0.0.

ifsrcaddr = address
Specifies the IP address of the source host interface that is used to send multicast packets. The default is 0.0.0.0.

mcadvert = address
Specifies the address for multicast messages. If you set mcadvert to something other than the default, the endpoints must also be changed to listen to the other address for multicast advertisements. The default is 224.0.1.118, which is the IANA-registered address for Tivoli multicast distributions.

mchigh = highest_address
Specifies the highest address to be used to send multicast data. When the server is ready to send multicast data, the server selects an address between mclow and mchigh to find an address that is available for multicast data traffic. If the first address checked is being used for sending multicast data, the address is incremented and the next address is monitored for activity. This continues until an available address or the value of mchigh is reached. The default is 224.2.255.255.

mclow = lowest_address
Specifies the lowest address to be used to send multicast data. When the server is ready to send multicast data, the server selects an address between mclow and mchigh to find an address that is available for multicast data traffic. If the first address checked is being used for sending multicast data, the address is incremented and the next address is monitored for activity. This continues until an available address or the value of mchigh is reached. The default is 224.2.128.0.

mc_netload = bytes_per_second
Specifies the maximum amount of network bandwidth, in bytes per second, that the repeater is allowed to use. The default is 500000.

mc_debug_level = trace_level
Specifies the trace level.
   0      Records no trace information
   1      Records exceptions only
   2      Records general trace information
   3      Records all implemented tracing
Trace levels are incremental. The trace logs are written locally on each repeater to $DBDIR\mcast.log. The default is 1.

**pollrtry**=`retry_count`

Specifies the maximum number of times that a multicast receiver will poll receivers to determine their final status. The default is 5.

**port**=`port_number`

Specifies the port number to use for multicast advertisements and multicast data. The default is 9499.

**rcvbufsz**=`size_in_bytes`

Specifies the size, in bytes, of the receive buffer of the UDP socket. The default is 250000.

**relrty**=`retry_count`

Specifies the number of times that a multicast receiver will broadcast the connection-release message to receivers. The default is 5.

**relwtout**=`time_in_milliseconds`

Specifies the time, in milliseconds, that the sender will wait for the receiver to release the connection after all data is transmitted. The default is 2000.

**repeat**=`count`

Specifies the number times that the server sends the same control packets. This can be increased if packet drop affects performance. The default is 2.

**returnIP**=`address`

Specifies the IP address of the server to which the receivers communicate. In satellite configurations, for example, the server-to-receiver traffic is a satellite link, and the receiver-to-server traffic is generally a telephone line; the return IP address will be different from the IP address of the source. The default is 0.0.0.0.

**sndbufsz**=`size_in_bytes`

Specifies the size, in bytes, of the send buffer of the UDP socket. The default is 250000.

**ttl**=`count`

Specifies the time-to-live integer. The integer indicates the number of times a packet can be forwarded by routers. When a packet is passed through a router, this integer is decremented; when it reaches zero, the packet is dropped. Specify a number larger that the number of routers between the multicast sender and receiver. The default is 5.

**Authorization**

*senior, super*

**See Also**

[wdepot] [wmdist] [wrpt]
wmdist

Configures repeaters and manages MDist 2 distributions.

Syntax

wmdist [-f] [-A] -B repeater_name

wmdist [-f] -c dist_id \ end point_id [end point_id...]  

wmdist [-f] [-h] -d dist_id [dist_id...]  

wmdist -D [debug_level]  

wmdist -e dist_id [-t ep_label] [-n node_type] [state...]  

wmdist -l repeater_name  

wmdist -j depot_directory...  

wmdist -k depot_directory...  

wmdist -l [-a] [-idist_id] [-v]  

wmdist -m dist_id [-t ep_label] [-n node_type] [state...]  

wmdist -M [TRUE | FALSE]  

wmdist -N [TRUE | FALSE]  

wmdist [-f] -p dist_id \ end point_id [end point_id...]  

wmdist -q dist_id  

wmdist [-f] -r dist_id \ end point_id [end point_id...]  

wmdist -R [rim_object]  

wmdist -s repeater_name [-C noprompt | nostart | nostop] [keyword=value...]  

wmdist -T [database_purge_interval]

Description

The wmdist command configures repeaters and manages MDist 2 distributions. To configure a repeater that uses the MDist service, see the wrpt command. If you have not configured an RDBMS Interface Module (RIM) object, you can only use the -s and -I options. For more information about MDist 2 and its services, see Tivoli Management Framework User’s Guide.

Options

- A Deletes all entries from the depot. Because you must delete the queue before the depot, always use the -A option with the -B option. Use this command only if instructed to do so by your Tivoli service provider.
-B repeater_name
   Deletes all entries from the queue. You might want to use this command in
   the following emergency situations:
   - If a gateway does not start or if a gateway crashes and you cannot
     restart it
   - If you issued a wmdist command with the -I option and observed data
     segments in the repeater queue from distributions that are no longer
     active.

Deleting entries from a repeater queue does not update the status database.
Use this command only if instructed to do so by your support provider.

-c dist_id | endpoint_id [endpoint_id...]  
Cancels an active distributions specified by dist_id, or cancels one or more
active distributions by target specified by endpoint_id.

-d dist_id [dist_id...]
Deletes one or more distributions from the MDist 2 database. The
distribution is specified by dist_id.

-D [debug_level]
Changes the level of log messages written by the distribution manager to
its log file, $DBDIR/distmgr.log. The debug_level option specifies a value
from 0 (least information) to 9 (most information). The default value is 0.
This file continues to increase in size unless you delete unwanted data and
maintain the amount of information that gets logged. Issuing this
command without a value prints the current value.

-e dist_id [-t ep_label] [-n node_type] [state...]
Lists endpoint status for a distribution. Suboptions are as follows:

-n node_type
   Filters output to only show repeaters or endpoints.

-t ep_label
   Specifies endpoints of the distribution. If endpoints are specified,
   the wmdist command lists status of these endpoints only.
   Otherwise, it lists statuses of all endpoints.

state
   Lists statuses for nodes (endpoints and repeaters) in the specified
   states. If not specified, the wmdist command lists statuses for all
   nodes. Supported states are as follows:
   - CANCELED
   - EXPIRED
   - FAILED
   - INTERRUPTED
   - PAUSED
   - RECEIVING
   - REJECTED
   - SENDING
   - STORED
   - SUCCESSFUL
   - UNAVAILABLE
   - WAITING

-f
Forces the operation and suppresses any confirmation prompt.

-h
Forces the removal from the database of distributions that have not
completed.
-I repeater_name
   Enables you to view detailed information about the distributions that the
   repeater is currently processing.

-j depot_directory...
   Add one or more alternative depot directories.

-k depot_directory...
   Removes one or more alternative depot directories.

-l [-i dist_id] [a] [-v]
   Lists distribution status. Options are as follows:
   -a Returns active distributions only.
   -i dist_id
      Specifies the distribution ID. When no distribution ID is specified,
      the command returns the status for all distributions.
   -v Returns all information about the status. If you do not specify the
      -v option, the command returns only the keyword value
      information.

-m dist_id [-t ep_label] [-n node_type] [state...]
   Lists the messages for a distribution. For a description of the suboptions,
   see the -e option.

-M [TRUE | FALSE]
   Controls whether an active MDist 2 distribution is accessible to an
   endpoint that migrates to another gateway. Specify TRUE to have the
   distribution accessible by the migrated endpoint; FALSE to not. If no option
   is provided, the current setting is displayed.

-N [TRUE | FALSE]
   Controls whether the distribution manager starts a distribution before
   validating database connectivity. Connectivity is required to register the
   distribution so that you can control the distribution using the Distribution
   Status console. If no option is provided, the current setting is displayed.
   The supported values are as follows:
   FALSE The distribution manager does not validate database connectivity
         before starting the distribution. An application-specific setting can
         override this setting. This is the default value.
   TRUE The distribution manager validates database connectivity before
        starting the distribution. An application-specific setting can
        override this setting.

-p dist_id [endpoint_id [endpoint_id...]]
   Pauses an active distribution specified by dist_id, or pauses one or more
   active distribution by targets specified by endpoint_id.

-q dist_id
   Displays the nodes associated with a given distribution in an indented
   format that indicates the route. Each node displayed is suffixed with its
   state.

-r dist_id [endpoint_id [endpoint_id...]]
   Resumes a paused distribution specified by dist_id, or resumes one or more
   paused distributions by target specified by endpoint_id.
-R [rim_object]
   Allows the user to change the RIM object used by the distribution manager to store status. The default value is mdist2. Issuing this command without a value prints the current value.

-s [-C [noprompt | nостart | nостop] repeater_name [keyword=value...]]
   Configures a repeater (specified by repeater_name) using one or more of the following keywords and values. If a value is not specified, the existing options for the specified repeater are displayed. When no keyword value pairs are listed, the command returns the configurations currently in use.

-C noprompt | nостart | nостop
   Specifies how a gateway repeater starts and stops the multicast receivers on the endpoints. The values are as follows:

   noprompt
   The repeater sends a message to start the multicast receiver without asking for confirmation. Depending on the number of endpoints assigned to the gateway, starting the multicast receiver on all the endpoints can take a long time. This option can be used when endpoint_multicast=TRUE.

   nостart
   The repeater does not send a message to start the multicast receiver on the endpoints. This option can be used when endpoint_multicast=TRUE.

   nостop
   The repeater does not send a shutdown message to stop the multicast receiver on the endpoints. The multicast receiver for each endpoint remains running. This option can be used when endpoint_multicast=FALSE.

Note: The –C option can only be used with the endpoint_multicast keyword. If specified with any other keyword, it is ignored.

repeater_name
   Specifies the label, object ID of the repeater, or one of the following options:

   all    Specifies all repeaters including the default setting.

   default    Specifies the default setting for a new repeater.

keyword=value
   The keywords and values are as follows:

   conn_retry_interval=seconds
   Specifies the frequency (in seconds) that unavailable or interrupted targets are retried.

   debug_level=number
   Controls which messages are written to log files for the managed node repeater ($DBDIR/rpt2log). Logging for the gateway repeater is controlled using the wgateway command with the debug_level option.

   default_multicast=TRUE | FALSE
   Specifies whether multicast is the the default mode for all MDist 2 distributions. This distribution mode can be
overridden by a Tivoli application. Specify TRUE to enable. Specify FALSE to not enable. The default is FALSE.

disk_max=\texttt{max\_size\_MB}  
Specifies the amount of disk space allocated to the repeater depot. Units are in megabytes (MB). If the disk_max number equals zero, no limit is enforced. This number should not exceed the size of the disk. Every distribution wmdist 288 flowing through a repeater is stored at least temporarily in the depot. The depot must be large enough to hold the largest distribution that you expect to distribute.

\textbf{Note:} If the new value is greater than the old value, changes are not effective until you restart the repeater.

disk_multicast=\texttt{TRUE \mid FALSE}  
When an application sends a multicast distribution, indicates whether the gateway uses multicast when distributing packages to its endpoints. Specify TRUE to use multicast. Specify FALSE to disable multicast and use unicast. A gateway can only multicast to its own endpoints. This keyword applies to gateway repeaters only. The default is FALSE.

\textbf{Notes:}
\begin{itemize}
  \item The keyword can be used in combination with the \texttt{–C} option.
  \item To set this keyword to \texttt{TRUE}, Java 1.3 for Tivoli and Tivoli Java Client Framework must be installed on this repeater.
\end{itemize}

deexecute_timeout=\texttt{seconds}  
Specifies the amount of time (in seconds) an endpoint method has to return results after it has received all that data from a distribution data. Some applications may be running scripts after receiving data but before returning results to the repeater.

fail_unavailable=\texttt{TRUE \mid FALSE}  
Specifies whether the gateway redistributes packages to endpoints that failed to receive a distribution. Specify FALSE to redistribute packages to endpoints that failed to receive a distribution. Specify TRUE to disable retry. This keyword applies to gateway repeaters only. An application-specific setting can override this setting. The default is FALSE.

\textbf{Note:} Unicast is used for the retry regardless of whether the original distribution was multicast or unicast.

max_sessions_high=\texttt{number}  
Specifies the maximum number of concurrent connections a repeater opens for high-priority distributions. These connections are shared among all active distributions. If the repeater runs out of high-priority connections, it tries to use a medium-priority connection.
**Note:** If the new value is greater than the old value, changes are not effective until you restart the repeater.

**max_sessions_low** = *number*

Specifies the maximum number of concurrent connections a repeater opens for low-priority distributions. This number must be one or greater. These connections are shared among all active distributions. If the repeater runs out of low-priority connections, it waits for an open connection to complete before opening additional connections.

**Note:** If the new value is greater than the old value, changes are not effective until you restart the repeater.

**max_sessions_medium** = *number*

Specifies the maximum number of concurrent connections a repeater opens for medium-priority distributions. These connections are shared among all active distributions. If the repeater runs out of medium-priority connections, it tries to use a low-priority connection.

**Note:** If the new value is greater than the old value, changes are not effective until you restart the repeater.

**mem_max** = *max_size_MB*

Specifies the amount of memory (in MB) that are used to buffer data being sent to targets. This improves performance by reducing the number of disk accesses to the depot. The memory is shared among all active distributions.

**Note:** If the new value is greater than the old value, changes are not effective until you restart the repeater.

**net_load** = *KB_per_second*

Specifies the maximum amount of network bandwidth (in kilobytes per second) that the repeater is allowed to use. If **slow_link** is set to TRUE, the network bandwidth is measured in bytes per second. This allocation is shared among all active distributions. This option is used with **target_netload**.

**notify_interval** = *minutes*

Specifies the frequency (in minutes) of status reporting. When the **notify_interval** has elapsed or the distribution has completed on all targets, the results are flushed. The results are sent to the application using MDist 2 and updated in the MDist 2 database.

**packet_size** = *number_KB*

Specifies the number of kilobytes written to the network during each send request. If **slow_link** is set to TRUE, specifies the number of bytes written to the network during each send request.
permanent_storage=TRUE | FALSE
Configures the repeater to be a depot. If set to TRUE, the depot retains segments marked for permanent storage after the distribution finishes. If set to FALSE, the segments are deleted from the depot after the distribution finishes.

repeater_multicast=TRUE | FALSE
When an application sends a multicast distribution, indicates whether the gateway uses multicast when distributing packages to other repeaters. Specify TRUE to use multicast. Specify FALSE to use unicast. The default is FALSE.

Note: To set this keyword to TRUE, Java 1.3 for Tivoli and Tivoli Java Client Framework must be installed on this repeater.

retry_ep_cutoff=seconds
Specifies the amount of time (in seconds) to continue retrying an unavailable or interrupted endpoint. Unavailable or interrupted repeaters are retried until the distribution deadline has been reached.

rpt_dir=path_name
Specifies the parent directory used to hold the \depot subdirectory and the states directory. The \depot directory contains all the segments stored in the database and must have enough free space to hold the value of the disk_max keyword. The states directory contains the database that holds the persistent state of the repeater queue.

send_timeout=seconds
Specifies the timeout (in seconds) used to detect a network or target failure while sending data. A target is allowed the number of seconds specified by the send_timeout option to receive each packet on the network. If a timeout occurs, the distribution remains in the repeater queue and a retry occurs in conn_retry_interval seconds. An application-specific setting can override this setting.

slow_link=TRUE | FALSE
Specifies whether the distribution is over links with bandwidths less than one kilobyte. If TRUE, the net_load, packet_size, and target_netload values are measured in bytes instead of kilobytes. The default is FALSE.

target_netload=KB_per_second
Specifies the maximum amount of network bandwidth (in kilobytes per second) that can be sent to an individual target. If slow_link is set to TRUE, the network bandwidth is measured in bytes per second. The default value of 0 disables this check. This command is used with the net_load keyword.

-T [database_purge_interval]
Sets the interval (in seconds) when completed distributions are deleted by the distribution manager from the RIM database. Setting this interval allows the distribution manager to delete completed distributions from the database after a distribution request is submitted. Although a purge
interval is defined, the completed distributions are not deleted unless the
defined interval has elapsed and a distribution request was submitted.
Issuing this command without a purge interval prints the current setting.
Setting the purge interval to -1 disables database purges. The default value
is -1.

Authorization
To cancel, delete, pause, or resume a distribution: senior or both Dist_control and
RIM_view.

To change the level of log messages written by the distribution manager to its log
file, configure the repeater with tuning options, change a RIM object to store status,
or set the interval at which completed distributions are removed from the RIM
database: senior.

To display the internal state of the specified repeater and list distributions in the
queue and connections in use, list endpoint status for a distribution, or display
nodes associated with a given distribution in an indented format that indicates the
route: senior or RIM_view.

To delete the contents of a queue: admin, senior, or super.

To control whether a distribution follows an endpoint that migrates to another
gateway while the distribution is active: senior.

To control whether the distribution manager validates database connectivity before
starting a distribution: senior.

Examples
1. The following example lists the status of all endpoints for the distribution with
the ID 11268691349760 that are in either the SUCCESSFUL or FAILED state:

   wmdist -e 11268691349760 SUCCESSFUL FAILED

   Name   Status   Start Time   End Time

2. The following example views the contents of the queue on repeater zyrous:

   wmdist -I zyrous

   Repeater: zyrous

   Jobs in SEND queue: 1
   Jobs in RECEIVE queue: 0

   ===Session Information ===
   Low: available = 40 used = 0
   Medium: available = 10 used = 0
   High: available = 5 used = 0

   ===Distribution Information ===

   External Id: 1185150392.113
   Internal Id: 1185150392.113
   Label: bobg20
   Priority: 3
   Application: mftp2debug

   Target: endpt123   State: PAUSED

3. The following example lists complete information about all active distributions:
wmdist

```
wmdist -l -a -v
```

4. The following example sets the value of `permanent_storage` to FALSE and sets
the value of `disk_max` to 51200 for all repeaters, including the default
configuration:
```
wmdist -s all permanent_storage=FALSE disk_max=51200
```

5. The following example allows you to view settings for repeater liliana:
```
wmdist -s liliana-gw
```

```
repeater_id: 1849216852.1.580
rpt_dir: /tmp/
permanent_storage: TRUE
max_sessions_high: 5
max_sessions_medium: 10
max_sessions_low: 40
disk_max: 500 (MB)
mem_max: 64 (MB)
send_timeout: 300 (secs)
execute_timeout: 600 (secs)
notify_interval: 30 (mins)
conn_retry_interval: 900 (secs)
retry_ep_cutoff: 7200 (secs)
net_load: 500 (KB/sec)
packet_size: 16 (KB)
target_netload: 0 (KB/sec)
debug_level: 3
repeater_multicast: FALSE
endpoint_multicast: FALSE
default_multicast: FALSE
SLOW_LINK: FALSE
```

Note that you can specify the label, object ID, or managed node ID of the
repeater.

6. The following example deletes all contents from both the zyrus repeater queue
and depot:
```
wmdist -B -A zyrus
```

Use this command only if instructed to by your support provider.

See Also
```
wmcast|wdepot|wep|wmdistgui|wrpl
```
wmdistgui

Starts the Distribution Status console from the managed node where invoked.

**Syntax**

```
wmdistgui
```

**Description**

The `wmdistgui` command start the Distribution Status console, which enables administrators to monitor and control distributions across a network.

Before running this command, the Distribution Status console software must be installed.

To run this command on a UNIX managed node, you must enable connections to the X Window System and also ensure that remote logins are enabled.

**Options**

None.

**Authorization**

You must have either the `senior` or `RIM_view` authorization role to view the status.
**wmemsize**

Reports the amount of physical memory installed on a managed node.

**Syntax**

```plaintext
wmemsize node_name
```

**Description**

The `wmemsize` command reports the amount of physical memory installed on a managed node. To run this command, the managed node must be available. The output is displayed to standard output and reports the number of megabytes of physical memory.

**Options**

```plaintext
node_name
```

- Specifies the name of the managed node to inspect.

**Authorization**

```plaintext
user, admin, senior, super
```

**Examples**

The following example displays the amount of memory installed on managed node ebbets.

```plaintext
wmemsize ebbets
```

```
64
```
wmerge

Performs a three-way file merge.

Syntax

wmerge [-L "label1" [-L "label3"]] [-p] [-q] "file1 file2 file3"

Description

The wmerge command incorporates all changes that lead from file2 to file3 into file1. When you specify the -p option, the result are written to standard output. Otherwise, the results are written into file1. This command is useful for combining separate changes to an original. For example, assume that file2 is the original, and both file1 and file3 are modifications to the original file. This command combines both sets of changes.

An overlap occurs if both modification files have changes in a common segment of lines. On a few older hosts where the diff3 command does not support the -E option, the wmerge command does not detect overlaps and merely supplies the changed lines from file3. On most hosts, if overlaps occur, the wmerge command outputs a message (unless the -q option is specified) and includes both alternatives in the result. The alternatives are delimited as follows:

<<<<<< "file1"
"lines in file1"
"========"
"lines in file3"
>>>>>> "file3"

If there are overlaps, the user should edit the result and delete one of the alternatives. When both -L options are specified, the labels are written to output instead of the names file1 and file3 in overlap reports.

Diagnostics

Exit status is 0 for no overlaps, 1 for some overlaps, and 2 for many overlaps.

Author


See Also

wco, wrcsmerge
wmrgaef

Merge custom dialogs from Tivoli Enterprise applications after upgrading.

Syntax

wmrgaef [–r resource...] –d path

Description

The wmrgaef command merges the custom dialogs from Tivoli Enterprise applications after an upgrade. Before running this command, you must first save the custom dialogs using the wcacher command.

The wmrgaef command attempts to merge the custom dialog with the upgraded dialog. The more similar the original and the upgraded dialogs, the easier it is to merge the custom dialog.

If the original and upgraded dialogs are completely different, it is possible that the merge could produce invalid Dialog Specification Language (DSL). In this case, that output from the wmrgaef command indicates that it was unable to merge that dialog. The command saves the original dialog, the upgraded dialog, and the merged attempt in the directory specified with the wcacher command.

Options

–d path

Specifies the path to the directory containing the custom dialogs. This is the same directory specified with the wcacher command; that is the parent path specified with the –d option and the subdirectory specified with the –s option.

–r resource

Specifies a resource type to merge. If you do not specify a resource type, the command reads all custom dialogs in the specified directory and tries to merge them.

Authorization

super

Examples

The following example merges custom dialogs saved by the wcacher command that are in the /tmp/aef/my.dir directory:

wmrgaef -d /tmp/aef/my.dir

See Also

wcacher
wmrgini

Merges groups and variables from one .INI file into another.

Syntax

`wmrgini destination_file source_file`

Description

The `wmrgini` command merges the contents of one .INI file in another. For each variable in the source file, the command creates or replaces an identical variable in destination file. This command should be run from an endpoint.

Options

`destination_file`

Specifies the name of the destination file.

`soure_file`

Specifies the name of the source file to merge.

Examples

The following example merges the `c:\temp\system.ini` file into the `c:\temp\win.ini` file:

`wmrgini c:\temp\win.ini c:\temp\system.ini`

See Also

`weditini`
wmv

Moves objects between collections.

Syntax

```
wmv [-I] label... collection
```

Description

The `wmv` command moves the specified objects from the current working collection into the specified collection. The command options can be full or partial label paths.

If you use this command to move an endpoint, use the `sync_gateways` option of the `wep` command to synchronize the endpoint data stored by the endpoint manager, gateways, and endpoints within the Tivoli region.

Options

- `-I` Ignores all failed suboperations, allowing the command to continue. This option is useful only when multiple labels are passed. This option allows a move process to fail for individual objects, but the command continues to the next object to be moved. Without this option, if a move process fails for an individual object, the command restores any objects already removed and then the command fails. The default is for the command to fail when a suboperation fails.

  - `collection` Specifies the label of the destination collection. This option can be a full label path (starting at the `/ collection), a partial label path (relative to the current working collection), or a simple name (to be found in the current working collection). The default is the current working collection. The linked objects become members of the selected collection.

  - `label` Specifies the labels of one or more objects to be moved. This option can be a full label path (starting at the `/ collection), partial label path (relative to the current working collection), or simple name (to be found in the current working collection).

Authorization

`admin`, `senior`, or `super` in the both policy regions as well as the appropriate global authorization role

Examples

The following example moves the ManagedNode object ceridwen from the lost-n-found collection to the desktop collection for administrator Root.

```
wmv /lost-n-found/ceridwen /Administrators/Root_ceridwen-region
```

See Also

`wep`, `wln`
wmvrim

Moves an RDBMS Interface Module (RIM) object to another managed node.

Syntax

wmvrim [–h host_name | –o host_oid] [–H db_home] [–I instance_home] [–t instance_name] rim_name

Description

The wmvrim command moves an existing RIM object to the specific managed node. When you use the –h or –o option, specify a managed node that is local to the Tivoli region where you enter the wmvrim command and where the existing RIM object resides. The –I option is required only for use with a DB2 database. You cannot change the vendor for a RIM object during a move operation. To change the vendor, you must delete the RIM object and create a new one.

Note: When you specify a path that contains a space, you must enclose the path name in quotation marks (" "). On Windows systems, you can also specify the MS-DOS path. For example, you can specify the path c:\Program Files\sqlib in one of the following ways:

• "c:\Program Files\sqlib"
• c:\progra~1\sqlib

Options

–h host_name
Specifies the host name of the managed node where the RIM object will reside. The managed node must be in the local Tivoli region. If you do not use this option or the –o option, the RIM object will not be moved.

–H db_home
Specifies the full path to the directory where the RDBMS server or client software is installed on the RIM host.

DB2
The directory where the DB2 software is installed. Equates to the value in the DB2DIR variable.

Oracle
The home directory. Equates to the ORACLE_HOME variable.

Sybase
The directory that contains the interfaces file. Equates to the SYBASE variable.

Microsoft SQL
The directory where Microsoft SQL Server is installed.

Informix
The directory where the Informix CLI client is installed. Equates to the value in the INFORMIXDIR variable.

Note: On Windows operating systems, use one of the following formats when specifying value for a path that contains spaces:

• "c:\Program Files\sqlib"
• c:\progra~1\sqlib
If you do not enclose the value within double quotation marks (") or include the tilde (~) in the path name, the parsing of the command fails.

`-I instance_home` (DB2 only)
Specifies the value of the INSTHOME variable. This value is the home directory where the database instance was created.

`-o host_oid`
Specifies the object ID of the managed node where the RIM object will reside. The managed node must be in the local Tivoli region. If you do not use this option or the `-h` option, the RIM object will not be moved.

`-t instance_name` (DB2 only)
Specifies the value of the DB2INSTANCE variable. If the RIM host is the same machine as the database server, this value must be the name of the server instance. If the RIM host is a different machine than the database server, this value must be the name of the client instance.

`rim_name`
Specifies the label of the existing RIM object.

**Authorization**

`senior` or `super` in the Tivoli region

**Examples**

1. The following example moves the inventory RIM object to managed node amon-sul:
   ```bash
   wmvrim -h amon-sul inventory
   ```
2. The following example moves the invdh2 RIM object to managed node amon-sul. The Oracle client software on the original RIM object is in the /tivoli/oracle directory. On managed node amon-sul, it is in the /oracle directory.
   ```bash
   wmvrim -h amon-sul -H /oracle invdh2
   ```

**See Also**

[wmvrim][wcrtrim][wdel]
wpatch

Installs a Tivoli Enterprise software patch.

Syntax

```
wpatch [-c source_dir] -i patch [-n] [-y] [install_variables]... managed_node...
```

Description

The `wpatch` command installs a Tivoli Enterprise software patch when invoked on the Tivoli server.

Options

`-c source_dir`

Specifies the complete path to the directory containing the installation image. For `source_dir`, you can specify either the fully qualified path (for example `/cdrom`) or the source host and the path in the format `source_host:path` (for example `oak:/cdrom`).

`-i patch`

Specifies the index file from which the patch is installed. Index files have an .IND extension. You can enter the file name with or without the file extension. For example, if the source directory contains the file TMF.IND, specifying either `-i TMF` or `-i TMF.IND` indicates the same file.

`-n`

Installs the patch on all managed nodes that do not currently have the patch installed. This option is ignored if a managed node is specified.

`-y`

Specifies that the installation should proceed without confirmation. By default, this command identifies the actions that must be taken to perform the installation and requests confirmation before continuing. Using this option, the `wpatch` command identifies the actions and performs the installation without requesting the confirmation.

`install_variables`

Specifies patch-specific installation variables. These generally take the form `@variable=value`. For details about the installation options for a specific product and its patches, refer to the product and patch documentation.

*Note:* Type the names of the installation options exactly as specified in the product documentation. Installation options are case sensitive.

`managed_node`

Specifies the managed node on which a Tivoli patch is installed. Multiple managed nodes can be specified. If no managed nodes are specified, the patch is installed on all managed nodes in the Tivoli management region. In most cases, this option is not specified.

Authorization

`senior` or `install_product`

See Also

`wclient`, `winstall`, `wserver`

Chapter 3. Commands alphabetically  289
**wping**

Attempts to contact the object dispatcher on a host.

**Syntax**

```
wping host_name [timeout]
```

**Description**

The `wping` command attempts to contact the object dispatcher on the specified host. If the object dispatcher responds, the following message is written to standard output:

```
object dispatcher on host_name is alive
```

If the object dispatcher does not respond, the following message is written to standard output:

```
no response from object dispatcher on host_name
```

**Options**

- `host_name`
  The name of the host to be contacted.

- `timeout`
  The number of seconds to wait for the host to reply before reporting that it is inactive.

**Authorization**

- user, admin, senior, super
wpopulate

Populates a profile from system files.

Syntax

wpopulate [–o] source profile

Description

The wpopulate command adds configuration information from a system managed by Tivoli Enterprise software to a configuration profile.

This command populates the identified profile based on the configuration found on the profile endpoint. The source option identifies the profile endpoint from which to populate. The type of information with which the profile is populated depends on the profile type.

If the –o option is specified, the wpopulate command overwrites the current profile contents. If the –o option is not specified, the contents from the source option are appended to the profile contents.

Options

–o Specifies to overwrite the contents of the specified profile.

source Specifies the name of the profile endpoint to use as the source for the profile.

profile Specifies the name of the profile to populate.

Authorization

senior, super

Examples

The following example populates the UnixUser UserProfile with the contents of the passwd file on managed node pinatubo. The content of the file is appended to the profile.

wpopulate @ManagedNode:pinatubo @UserProfile:UnixUser

See Also

wcrtpfr, wcrtpfrmgr, wdistrib, wgetprf, wgetsub, wlssub, wsub, wunsub, wxterm
wputeppol

Replaces an endpoint policy script that has been modified.

Syntax

wputeppol policy_script

Description

The wputeppol command replaces an endpoint policy script. After the script has been modified, this command saves the changes by writing the script to the Tivoli database. Script contents must be entered through standard input.

Options

policy_script

Specifies the name of the policy script to be replaced.

Authorization

senior

Examples

The following example saves changes made to the login policy script by writing the policy script to the database:

wputeppol login_policy < login_policy

See Also

wgeteppol
wputpolm

Replaces the body of a policy method.

**Syntax**

```
wputpolm [-C | -c value] [-d | -v] class name policy
wputpolm [-d | -v] [-F | -N] [-n | -C | -c value] [args='a1,...'] profile policy
``` 

**Description**

The `wputpolm` command replaces the body of the specified policy method. There are two forms for the `wputpolm` command: the form for replacing a policy method of a traditional Tivoli managed resource, and the form for replacing the value of a policy on a profile.

In the first form, the `class` option specifies the type of resource, and `name` specifies the label of the policy object on which to act. If the `-d` option is specified, `name` specifies a policy default object name and default policy for the resource.

This form of the command can be used to define the policy method in one of two ways. It can read from its standard input the body of the policy method. In this form, `wputpolm` is intended for defining methods that are implemented as shell scripts. However, this form also works on binary files and accepts an executable program from the standard input. The command can also define a policy method that has a constant value. In this form, it is not necessary to write a shell script. These constant methods require less storage space and execute more quickly than shell-script methods. The command can read the constant return value either from the command line, using the `-c` option, or from standard input, if the `-C` option is specified.

In the second form of the command, the `profile` option specifies the profile name, and the `policy` option specifies the individual attribute whose policy to set, as returned by the `wlspolm` command. The `-C`, `-c`, `-d`, and `-v` options behave identically for profiles and managed resources and can be used to install constant- or script-valued policies on a profile. When running `wputpolm` on a profile, the `-n` option is also available. The `-n` option indicates none. Used with `-d` for a default policy, `-n` means that there is no default value for the individual attribute. Used in conjunction with `-v` for a validation policy, `-n` indicates that any value is valid for the policy.

The `-F` and `-N` options are used to set the policy to be fixed or not fixed, respectively. If neither flag is specified, the fixed status of the policy is left unchanged.

The optional `args` list is valid only for script-valued policies, and identifies the input options to pass to the policy script. For any input options of the form `$attribute`, where `attribute` is the name of an attribute of the resource type, the attribute value is passed as input to the script.

**Note:** Adding a policy method does not place a notice in any notice group, because policy is dependent on the type of managed resource to which the policy method applies.
Input
If the –C option is present, this command reads from the command standard input the constant return value to be set for the policy method.

If the –c option is present, this command ignores its standard input.

If the –C and –c options are both omitted, this command reads the body of the method (usually a shell script) from the command standard input.

Options
-c value
Specifies that the policy method is defined always to return the specified constant value. value is an alphanumeric American National Standard Code for Information Interchange (ASCII) string. Numeric values are read and stored as strings. If this option and the –C option are omitted, this command reads the method body from standard input.

-C
Specifies that the policy method is defined always to return the specified constant value read from this command standard input. If this option and the –c option are omitted, this command reads the method body from standard input.

-d
Specifies that the method is a default policy.

-F
Specifies that the policy is fixed and the subscribers cannot override it.

-n
Specifies that either there is no default value for the attribute of the policy or any value is valid for the policy.

-N
Specifies that the policy is not fixed. After distribution, subscribers can override the policy.

-v
Specifies that the method is a validation policy.

args='a1,...'
Specifies additional arguments to the method.

class
Specifies the managed resource type to which the policy is assigned.

name
Specifies the name of the managed resource.

policy
Specifies the name of the method whose body is to be defined or replaced.

profile
Specifies the profile to which the policy is assigned.

Authorization
For the first form of the command, super and policy.

For the second form of the command, senior or super.

Examples
1. For non-profile use, the following example installs a new policy script from the pm_val_subscribers validation script for the Restricted policy validation object for the ProfileManager type. The script is read from standard input.
   wputpolm -v ProfileManager Restricted pm_val_subscribers < new_script

2. For non-profile use, the following example sets the policy for the pm_val_subscribers validation script to be the constant TRUE, which means that all subscribers are accepted:
   wputpolm -v -c TRUE ProfileManager Restricted pm_val_subscribers
3. For profile use, the following example installs a new default policy script for user ID (UID) generation for the Engineering user profile. The policy is fixed so that subscribers cannot override it. The script takes the user real name and login name as its arguments. The script is read from standard input.

```bash
wputpolm -d -F args='real_name,login_name' \
@UserProfile:Engineering uid < new_script
```

**See Also**

- wchkpol
- wcrtpol
- wcrtp
- wdelpol
- wdelpr
- wgetdfpol
- wgetpolm
- wgetpol
- wlspol
- wlspolm
wpwd

Displays the current working collection.

**Syntax**

wpwd [–o]

**Description**

The `wpwd` command displays the label of the administrator current working collection for the current parent process ID. Each administrator has a separate current working collection associated with each parent process ID.

**Options**

–o Displays the object ID of the collection. When not specified, the command displays the collection label.

**Authorization**

user, admin, senior, super

**See Also**

wcd
**wrcs**

Changes Revision Control System (RCS) file attributes.

**Syntax**

```
wrcs [options] file...
```

**Description**

The `wrcs` command creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For `wrcs` to work, the caller login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the super user, or the `-i` option is present.

Paths matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in the `wci` command. Revision numbers use the syntax described in the `wci` command.

**Options**

- `-a logins`
  Appends the login names appearing in the comma-separated list `logins` to the access list of the RCS file.

- `-A old_file`
  Appends the access list of `old_file` to the access list of the RCS file.

- `-b [rev]`
  Sets the default branch to `rev`. If `rev` is omitted, the default branch is reset to the (dynamically) highest branch on the trunk.

- `-c string`
  Sets the comment leader to string. An initial the `wci` command, or a `wrcs` `-i` command without the `-c` option, guesses the comment leader from the suffix of the working file name. This option is obsolescent, because RCS typically uses the prefix of the preceding `$Log$` line when inserting log lines during checkout (see the `wco` command). However, older versions of RCS use the comment leader instead of the prefix of the `$Log$` line, so if you plan to access a file with both old and new versions of RCS, make sure that its comment leader matches its `$Log$` line prefix.

- `-e [logins]`
  Erases the login names appearing in the comma-separated list `logins` from the access list of the RCS file. If `logins` is omitted, erases the entire access list.

- `-i`
  Creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, tries to place it first into the `.RCS` subdirectory, and then into the current directory. If the RCS file already exists, prints an error message.

- `-I`
  Runs interactively, even if the standard input is not a terminal.

- `-k subst`
  Sets the default keyword substitution to `subst`. The effect of keyword substitution is described in the `wco`. Giving an explicit `-k` option to the `wco` command, the `wrcsdiff` command, and the `wrcsmmerge` command overrides this default. Do not use the `wrcs -kv` command, because the `-kv`
option is incompatible with the \texttt{wco \textasciitilde \textasciitilde} command. Use the \texttt{wrcs \textasciitilde-k kv} command to restore the usual default keyword substitution.

\texttt{-l [rev]}

Locks the revision with number \texttt{rev}. If a branch is given, locks the latest revision on that branch. If \texttt{rev} is omitted, locks the latest revision on the default branch. Locking prevents overlapping changes. A lock is removed with the \texttt{wci} or \texttt{wrcs \textasciitilde-u} command.

\texttt{-L}

Sets locking to \textit{strict}. Strict locking means that the owner of an RCS file is not exempt from locking for check in. This option should be used for files that are shared.

\texttt{-m rev:msg}

Replaces the revision log message for \texttt{rev} with \texttt{msg}.

\texttt{-M}

Does not send mail when breaking a lock. This option is not meant for casual use; it is meant for programs that warn users by other means and invoke the \texttt{wrcs \textasciitilde-u} only as a low-level lock-breaking operation.

\texttt{-n name:[:rev]}

Associates the symbolic name \texttt{name} with the branch or revision \texttt{rev}. Deletes the symbolic name if both the colon (:) and \texttt{rev} options are omitted; otherwise, prints an error message if \texttt{name} is already associated with another number. If \texttt{rev} is symbolic, it is expanded before association. A \texttt{rev} consisting of a branch number followed by a period (.) stands for the current latest revision in the branch. A colon with an empty \texttt{rev} stands for the current latest revision on the default branch, typically the trunk. For example, \texttt{wrcs \textasciitilde-n name: RCS/*} associates \texttt{name} with the current latest revision of all the named RCS files; this contrasts with \texttt{wrcs \textasciitilde-n name: RCS/*}, which associates \texttt{name} with the revision numbers extracted from keyword strings in the corresponding working files.

\texttt{-N name:[:rev]}

Acts like the \texttt{-n} option, except overrides any previous assignment of \texttt{name}.

\texttt{-o range}

Deletes (outdates) the revisions given by \texttt{range}. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form \texttt{rev1:rev2} means revisions \texttt{rev1} to \texttt{rev2} on the same branch, \texttt{:rev} means from the beginning of the branch containing \texttt{rev} up to and including \texttt{rev}, and \texttt{rev:} means from revision \texttt{rev} to the end of the branch containing \texttt{rev}. None of the outdated revisions may have branches or locks.

\texttt{-q}

Runs quietly; does not print diagnostics.

\texttt{-s state[:rev]}

Sets the state attribute of the revision \texttt{rev} to \texttt{state}. If \texttt{rev} is a branch number, assumes the latest revision on that branch. If \texttt{rev} is omitted, assumes the latest revision on the default branch. Any identifier is acceptable for \texttt{state}. A useful set of states is \texttt{Exp} (for experimental), \texttt{Stab} (for stable), and \texttt{Rel} (for released). By default, the \texttt{wci} command sets the state of a revision to \texttt{Exp}.

\texttt{-t [file]}

Writes descriptive text from the contents of the named \texttt{file} into the RCS file, deleting the existing text. The \texttt{file} path name may not begin with a dash (\texttt{-}). If \texttt{file} is omitted, obtains the text from standard input, terminated by end-of-file or by a line containing a period (\texttt{.}) by itself. Prompts for the text
if interaction is possible; see the –I option. With the –i option, descriptive
text is obtained even if the –t option is not given.

–t - string
   Writes descriptive text from string into the RCS file, deleting the existing
text.

–T
   Preserves the modification time on the RCS file unless a revision is
removed. This option can suppress extensive recompilation caused by a
make dependency of some copy of the working file on the RCS file. Use
this option with care; it can suppress recompilation even when it is
needed, that is, when a change to the RCS file would mean a change to
keyword strings in the working file.

–u [rev]
   Unlocks the revision with number rev. If a branch is given, unlocks the
latest revision on that branch. If rev is omitted, removes the latest lock held
by the caller. Typically, only the locker of a revision may unlock it.
Somebody else unlocking a revision breaks the lock. This causes a mail
message to be sent to the original locker. The message contains a
commentary solicited from the breaker. The commentary is terminated by
end-of-file or by a line containing a period (.) by itself.

–U
   Sets locking to non-strict. Non-strict locking means that the owner of a file
need not lock a revision for check in. This option should not be used for
files that are shared. Whether default locking is strict is determined by
your system administrator, but it is typically strict.

–V
   Prints RCS version number.

–V n
   Emulates RCS Version n. See the wco command for details.

–x suffixes
   Uses suffixes to characterize RCS files. See the wci command for details.

–z zone
   Uses zone as the default time zone. This option has no effect; it is present
for compatibility with other RCS commands.

At least one explicit option must be given to ensure compatibility with future
planned extensions to the wrcs command.

Compatibility
The –b rev option generates an RCS file that cannot be parsed by RCS version 3 or
earlier.

The –k subst options (except –k kv) generate an RCS file that cannot be parsed by
RCS version 4 or earlier.

Use wrcs –V n to make an RCS file acceptable to RCS version n by discarding
information that would confuse version n.

RCS version 5.5 and earlier does not support the –x option and requires a ,v suffix
on an RCS path name.

Diagnostics
The RCS path name and the revisions outdated are written to the diagnostic
output. The exit status is zero if and only if all operations were successful.
**Files**

The `wrcs` command accesses files much as the `wci` command, except that the `wrcs` command uses the effective user for all accesses, it does not write the working file or its directory, and it does not read the working file unless a revision number of `$` is specified.

**Environment Variables**

**RCSINIT**

Options prepended to the option list, separated by spaces. See the `wci` command for details.

**Defects**

A catastrophe (for example, a system crash) can cause RCS to leave behind a semaphore file that causes later invocations of RCS to claim that the RCS file is in use. To fix this, remove the semaphore file. A semaphore file name typically begins with a comma (,) or ends with an underscore (_).

The separator for revision ranges in the `–o` option used to be a dash (–) instead of a colon (:) , but this leads to confusion when symbolic names contain dashes. For backward compatibility, the `wrcs –o` command still supports the dash separator, but it warns about this obsolete use.

Symbolic names need not refer to existing revisions or branches. For example, the `–o` option does not remove symbolic names for the outdated revisions; you must use `–n` to remove the names.

**Author**

Author: Walter F. Tichy. Revision Number: 5.13; Release Date: 1995/06/05.

**See Also**

wrcsdiff

Compares Revision Control System (RCS) revisions.

Syntax

\texttt{wrcsdiff [-k subst] [-q] [-r rev1 [-r rev2]] [-T] [-V n] [-x suffixes] [-z zone] [diff_options] file...}

Description

The \texttt{wrcsdiff} command runs the \texttt{diff} command to compare two revisions of each RCS file given.

Paths matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in the \texttt{wci} command. The \texttt{-q} option suppresses diagnostic output. Zero, one, or two revisions can be specified with the \texttt{-r} option. The \texttt{-k} option affects keyword substitution when extracting revisions, as described in the \texttt{wco} command; for example, \texttt{-k k-r1.1-r1.2} ignores differences in keyword values when comparing revisions 1.1 and 1.2. To avoid excess output from locker name substitution, \texttt{-k kv} is assumed if (1) at most one revision option is given, (2) no \texttt{-k} option is given, (3) \texttt{-k kv} is the default keyword substitution, and (4) the mode of the working file would be produced by the \texttt{wco -l} command. See the \texttt{wco} command for details about the \texttt{-T}, \texttt{-V}, \texttt{-x}, and \texttt{-z} options. Otherwise, all options of the \texttt{diff} command that apply to regular files are accepted, with the same meaning as for the \texttt{diff}. If both \texttt{rev1} and \texttt{rev2} are omitted, \texttt{wrcsdiff} compares the latest revision on the default branch (by default the trunk) with the contents of the corresponding working file. This is useful for determining what you changed since the last check in. If \texttt{rev1} is given, but \texttt{rev2} is omitted, \texttt{wrcsdiff} compares revision \texttt{rev1} of the RCS file with the contents of the corresponding working file. If both \texttt{rev1} and \texttt{rev2} are given, \texttt{wrcsdiff} compares revisions \texttt{rev1} and \texttt{rev2} of the RCS file. Both \texttt{rev1} and \texttt{rev2} might be given numerically or symbolically.

Diagnostics

Exit status is 0 for no differences during any comparison, 1 for some differences, 2 for many differences.

Environment Variables

\texttt{RCSINIT}

Options prepended to the option list, separated by spaces. See the \texttt{wci} command for details.

Examples

The following command compares the latest revision on the default branch of the RCS file to the contents of the working file \texttt{f.c}:

\texttt{wrcsdiff f.c}

Author

See Also

**wrcsmerge**

Merges Revision Control System (RCS) revisions.

**Syntax**

`wrcsmerge [options] file`

**Description**

The `wrcsmerge` command incorporates the changes between two revisions of an RCS file into the corresponding working file.

Paths matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in the `wci` command.

At least one revision must be specified with one of the options described below, usually `-r`. At most two revisions may be specified. If only one revision is specified, the latest revision on the default branch (typically the highest branch on the trunk) is assumed for the second revision. Revisions may be specified numerically or symbolically.

`wrcsmerge` prints a warning if there are overlaps and delimits the overlapping regions. The command is useful for incorporating changes into a checked-out revision.

**Options**

- `-k subst`  
  Uses `subst` style keyword substitution. See the `wco` command for details. For example, `-k k-r1.1-r1.2` ignores differences in keyword values when merging the changes from 1.1 to 1.2.

- `-p [rev]`  
  Sends the result to standard output instead of overwriting the working file.

- `-q [rev]`  
  Runs quietly; does not print diagnostics.

- `-r [rev]`  
  Merges with respect to revision `rev`. Here an empty `rev` stands for the latest revision on the default branch, typically the head.

- `-V n`  
  Emulates RCS version `n`. See the `wco` command for details.

- `-x suffixes`  
  Uses `suffixes` to characterize RCS files. See the `wci` command for details.

**Diagnostics**

Exit status is 0 for no overlaps, 1 for some overlaps, 2 for many overlaps.

**Environment Variables**

- `RCSINIT`  
  Options prepended to the option list, separated by spaces. See the `wci` command for details.
Examples

1. Suppose you have released revision 2.8 of the f.c file. Assume furthermore that after you complete an unreleased revision 3.4, you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into the f.c file and enter the following command:

   wrcsmerge -p -r2.8 -r3.4 f.c >f.merged.c

   Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and enter wco -j:

   wci -r2.8.1.1 f.c
   wco -r3.4 -j2.8:2.8.1.1 f.c

2. As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in the f.c file:

   wrcsmerge -r2.8 -r2.4 f.c

   Note the order of the options and that f.c file is overwritten.

Author


See Also

wrefresh

Refreshes a Tivoli collection window.

Syntax

    wrefresh collection

Description

The wrefresh command refreshes the collection window of the specified collection.

Options

    collection

        Specifies which collection window is refreshed. To refresh the Administrators window, use the following format:

            /Administrators/administrator_name

        To refresh other collection windows, use one of the following formats:

            /Regions/top_level_region_name/subregion_name

            @NisDomain:domain_name

Authorization

    user

Examples

1. The following example refreshes all instances of the desktop for administrator Callahan. If multiple Callahan desktops are open, all are refreshed.

    wrefresh /Administrators/Callahan

2. The following example refreshes all instances of the New York policy region:

    wrefresh @PolicyRegions:NewYork
wregister

Registers a new resource instance with the name registry or enables exchangeability of resources between connected Tivoli regions.

Syntax

wregister -i [-fn] -r resource_type
wregister -i [-fr | -fx] -r resource_type
wregister [-i [-fn] -r resource_type] name object
wregister -u [-r resource_type] name

Description

The wregister command registers a new resource instance with the Tivoli name registry or enables exchangeability of resources between connected Tivoli regions.

When used to mark an exchangeable resource as a remote resource type, the resource can no longer be exchanged between connected Tivoli regions. When you run a wlookup command against remote resource types, the output returns a list of matching resource types by Tivoli region. Basically, the sort order is different after a resource type is made remote.

When used to create a new resource instance, the command optionally initializes the cache for a new resource type. If -r is not specified, the default resource type is distinguished.

Options

- fn Specifies that the resource type being created should be nonexchangeable. Nonexchangeable resource types cannot be updated between connected Tivoli regions.
- fr Specifies that an exchangeable resource type be made a remote resource type. When an exchangeable resource type is remote, there is no need to update it between connected Tivoli regions.
- fx Specifies that a remote resource type be returned to an exchangeable resource type. These resource types can be updated between connected Tivoli regions.
- i Initializes the resource cache. If this option is not specified and the specified resource type does not already exist in the cache, wregister generates an error.
- r resource_type Specifies the resource type of the resource to be registered. If omitted, the default resource type is distinguished.
- u Removes a resource from a resource type.

name Specifies the name under which the resource is to be registered.
object Specifies the object reference of the resource.
Authorization
senior, super

Examples
1. The following example adds a new resource type, MyResource, to the name registry. The new resource type is nonexchangeable.
   \texttt{wregister -i -fn -r MyResource}
2. The following example adds a resource named mylabel to MyResource. The object ID for mylabel is 400004.34.26.
   \texttt{wregister -r MyResource mylabel 400004.34.26}
3. The following example removes the resource mylabel from MyResource:
   \texttt{wregister -r MyResource -u mylabel}
4. The following example adds a new resource type, YourResource, and adds a resource named yourlabel to YourResource. The object ID of yourlabel is 400005.35.37.
   \texttt{wregister -i -r YourResource yourlabel 400005.35.37}

See Also
\texttt{wlookup wupdate}
**wrestart**

Initiates a system restart and optional restart. (Windows only)

**Syntax**

```
wrestart [-b] [-c] [-f] [-t timeout_value] [-m "confirm_message"]
```

**Description**

The `wrestart` command initiates a system restart and optional restart.

**Options**

- `-b` Restarts the system after shutdown.
- `-c` Prompts the user for confirmation before restarting the system.
- `-f` Forces a restart in spite of unsaved changes in other applications. This option does not cause applications to prompt users to save their changes.
- `-m "confirm_message"` Specifies the confirmation message that is displayed.
- `-t timeout_value` Specifies the number of seconds that the command waits for user confirmation before initiating a restart.

**Examples**

1. To prompt the user for confirmation before restarting the system, enter the following command:
   ```
wrestart -c
   ```

2. To display a warning message before restarting the system in 60 seconds, enter the following command:
   ```
wrestart -m "Warning: The system will reboot in 60 seconds." -t 60
   ```
wrimtest

Verifies connectivity and functionality of an RDBMS Interface Module (RIM) object.

Syntax

\texttt{wrimtest \textit{--l \textit{RIM\_object\_label}}}

Description

The \texttt{wrimtest} command is an interactive command-line utility that enables you to connect to a specified database and run RIM methods.

After the command establishes the requested connection, you can enter one of the following options to test functionality:

- \texttt{c} Commits a transaction.
- \texttt{d} Deletes a row from the database.
- \texttt{e} Executes a Structured Query Language (SQL) statement.
- \texttt{g} Retrieves rows from the database.
- \texttt{i} Inserts a row into the database.
- \texttt{r} Cancels a transaction.
- \texttt{u} Updates rows in the database.
- \texttt{?} Prints a list of command options.
- \texttt{x} Exits the utility.

Options

\texttt{--l \textit{RIM\_object\_label}}

Specifies which RIM object to test. For a list of available RIM objects, use the \texttt{wlookup --ar RIM} command.

Authorization

\texttt{rim\_view, rim\_update}

See Also

\texttt{wrimtrace}
wrimtrace

Enables or disables tracing for RDBMS Interface Module (RIM) objects.

Syntax

```
  wrimtrace  RIM_object_label [trace_levels]
```

Description

The `wrimtrace` command enables or disables tracing for RIM objects. The contents of the Inter-Object Messaging (IOM) packets passed between the RIM object and client program and the native RDBMS errors that are printed to the RIM log file, which is located in `/tmp/rim_db_log` directory. You can locate and change the default location of the RIM log file by completing the following steps:

1. Run the following command:
   ```
   odadmin environ get > env.out
   ```
2. Edit the env.out file and add the following:
   ```
   RIM_DB_LOG=/tivoli/rim/rim_db_log
   ```
3. Run the following command:
   ```
   odadmin environ set < env.out
   ```

Running the `wrimtrace` command, without a trace level option (INFORMATION, ERROR, or TRACE_OFF), prints the current trace level to standard output. Tracing is intended for debugging purposes. If enabled for extended periods of time, tracing can decrease performance and slow the RIM processing.

**Note:** When you change the RIM tracing level, you must locate and stop the appropriate `RIM_database_AGENT` process on the machine, where `database` is one of the following:

- For DB2
  ```
  DB2
  ```
- For Informix
  ```
  Informix
  ```
- For Microsoft SQL Server
  ```
  MS_SQL
  ```
- For Oracle
  ```
  Oracle
  ```
- For Sybase
  ```
  Sybase
  ```

Options

- **RIM_object_label**
  Specifies the RIM object you want to trace.

- **trace_levels**
  Specifies the appropriate trace levels:
  - **ERROR**
    Prints RDBMS errors to the RIM log file.
  - **INFORMATION**
    Prints the contents of IOM packets to the RIM log file.
TRACE_OFF
Turns tracing off.

Examples
1. The following example prints the current trace level of the inventory RIM object:
   `wrimtrace inventory`
2. The following example prints IOM packet information to the RIM log file:
   `wrimtrace inventory INFORMATION`
3. The following example prints IOM packet information and RDBMS errors to the RIM log file:
   `wrimtrace inventory "INFORMATION|ERROR"`
4. The following example turns RIM tracing off:
   `wrimtrace inventory TRACE_OFF`

See Also
`wrimtest`
wrlog

Prints log messages and other information about Revision Control System (RCS) files.

Syntax

```
wrlog [options] file...
```

Description

The `wrlog` command prints information about RCS files.

Paths matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in the `wci` command.

`wrlog` prints the following information for each RCS file: RCS path name, working path name, head (for example, the number of the latest revision on the trunk), default branch, access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, `wrlog` prints revision number, author, date and time, state, number of lines added or deleted (with respect to the previous revision), locker of the revision (if any), and log message. All times are displayed in Coordinated Universal Time (UTC). Without options, `wrlog` prints complete information. The following options restrict this output.

Options

- `-b`  Prints information about the revisions on the default branch, typically the highest branch on the trunk.
- `-d dates`  Prints information about revisions with a check-in date and time in the ranges given by the semicolon-separated list of dates. A range of the form `d1<d2` or `d2>d1` selects the revisions that were deposited between `d1` and `d2` inclusive. A range of the form `<d` or `d>` selects all revisions dated `d` or earlier. A range of the form `d<` or `>d` selects all revisions dated `d` or later. A range of the form `d` selects the single, latest revision dated `d` or earlier. The date and time strings `d`, `d1`, and `d2` are in the free format explained in `wco`. Quoting is typically necessary, especially for `<` and `>`. Note that the separator is a semicolon.
- `-h`  Prints only the RCS path name, working path name, head, default branch, access list, locks, symbolic names, and suffix.
- `-l [lockers]`  Prints information about locked revisions only. In addition, if the comma-separated list `lockers` of login names is given, ignores all locks other than those held by `lockers`. For example, `wrlog -l -R -l wft RCS/*` prints the name of RCS files locked by the user `wft`.
- `-L`  Ignores RCS files that have no locks set. This is convenient in combination with `-h`, `-l`, and `-R`.
- `-r [revisions]`  Prints information about revisions given in the comma-separated list, `revisions`, of revisions and ranges. A range of the form `rev1:rev2` means revisions `rev1` to `rev2` on the same branch; `:rev` means revisions from the
beginning of the branch up to and including \textit{rev}, and \textit{rev}: means revisions starting with \textit{rev} to the end of the branch containing \textit{rev}. An option that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range. A branch followed by a perion (.) means the latest revision in that branch. A bare \texttt{r} without \textit{revisions} means the latest revision on the default branch, typically the trunk.

\texttt{–R}  
Prints only the name of the RCS file. This is convenient for translating a working path name into an RCS path name.

\texttt{–s states}  
Prints information about revisions whose state attributes match one of the states given in the comma-separated list \textit{states}.

\texttt{–t}  
Prints the same information as \texttt{–h}, timeout plus the descriptive text.

\texttt{–V n}  
Emulates RCS version \textit{n} when generating logs. See the \texttt{wco} command for more details.

\texttt{–w [logins]}  
Prints information about revisions checked in by users with login names appearing in the comma-separated list \textit{logins}. If \textit{logins} is omitted, the user login is assumed.

\texttt{–x suffixes}  
Uses \textit{suffixes} to characterize RCS files. See the \texttt{wci} command for details.

The \texttt{wrlog} command prints the intersection of the revisions selected with the options \texttt{–d, –l, –s, and –w}, intersected with the union of the revisions selected by \texttt{–b} and \texttt{–r}.

\textbf{Diagnostics}  
The exit status is zero if, and only if, all operations were successful.

\textbf{Environment Variables}  
\texttt{RCSINIT}  
Options prepended to the option list, separated by spaces. See the \texttt{wci} command for details.

\textbf{Examples}  
1. The following command prints the names of all RCS files in the \texttt{\RCS} subdirectory that have locks:  
\texttt{wrlog \-L \-R RCS/*}

2. The following command prints the headers of those files:  
\texttt{wrlog \-L \-h RCS/*}

3. The following command prints the headers plus the log messages of the locked revisions:  
\texttt{wrlog \-L \-l RCS/*}

4. The following command prints complete information:  
\texttt{wrlog RCS/*}
Defects

The separator for revision ranges in the –r option used to be dash (–) instead of a colon (:), but this leads to confusion when symbolic names contain dashes. For backward compatibility, the `wrlog –r` still supports the old dash separator, but it warns about this obsolete use.

Author


See Also

wrm

Removes objects from a collection.

**Syntax**

wrm [-I] label...

**Description**

The *wrm* command removes the specified objects from the collection. You can specify the object to be removed by label or label path. This command removes only references to objects; it does not delete the objects themselves.

**Options**

- **-I** Ignores all failed suboperations, allowing the command to continue. This option is useful only when multiple labels are passed to the command. This option allows a removal operation to fail for individual objects, but the command continues to the next object to be removed. Without this option, if a remove operation fails for an individual object, the command restores any objects already removed, and then the command terminates with an error. The default is for the command to fail when the suboperation fails.

- **label** Specifies the labels or label paths of the objects to be removed. Objects are specified similarly to UNIX file names. If a full or relative label path is specified, the object is removed from the collection that the label path names; that is, the object is removed from the collection named by the label path less its final component. If only the unadorned object label is specified, the object is removed from the current working collection.

**Authorization**

admin, senior, super

**Examples**

The following example removes the reference for the ManagedNode object ceridwen from the Administrators collection:

wrm /Administrators/Root_ceridwen-region/ceridwen

**See Also**

[wdel]
wrmnode

Removes a managed node from a Tivoli environment.

Syntax

\texttt{wrmnode [\textbf{-f}] node\_name [\textbf{-d} dispatcher\_number] [node\_name [\textbf{-d} dispatcher\_number]]...}

Description

The \texttt{wrmnode} command removes the specified managed node from the Tivoli database. Although the Tivoli server is a managed node, this command cannot be used to remove the Tivoli server. For a UNIX managed node, this command shuts down the dispatcher, and then removes the managed node from the Tivoli database.

Managed nodes are specified by name. If a managed node is damaged, it might be necessary to provide the dispatcher number. To obtain the dispatcher number, use the \texttt{odadmin odlist} command.

After using removing the managed node, you might want to use the \texttt{wchkdb} command to verify the database to ensure that there are no references to the removed managed node.

Options

\textbf{-d.dispatcher\_number}

Shuts down the specified dispatcher and removes a damaged managed node.

\textbf{-f}

Performs all requested removals without requiring confirmation from the user.

\textit{node\_name}

Specifies the name of the managed node to be removed.

Authorization

\texttt{install\_client, super}

Notes

This command does not delete Tivoli Management Framework executables, databases, or other files from the managed node.

Examples

1. The following example removes managed node sherman from the Tivoli database:

\begin{verbatim}
wrmnode sherman
\end{verbatim}

2. The following example removes a partially installed or partially removed managed node. Because the managed node is not fully known, the dispatcher number must be provided. To determine the dispatcher number, enter the following command:

\begin{verbatim}
Tivoli Management Framework: Reference Manual
\end{verbatim}

316
odadmin odlist

Region Disp Flags Port IPaddr Hostname(s)
23 2323 1 ct- 94 146.84.25.15 ceridwen,ceridwen.tivoli.com
2 2 2 -t- 94 146.84.29.12 elcap,elcap.tivoli.com

To remove this partially managed node, enter the following command:

wrmnode elcap -d 2

See Also

odadmin wbkupdb wchkdb wclient winstall wpatch wserver
**wrplblk**

Replaces a block of statements in a file. This command should be run from an endpoint.

**Syntax**

```
wrplblk [-r] -s "start_string" -e "end_string" [-o output_file] [-i "replace_string" | @file_name] file_name
```

**Description**

The `wrplblk` command replaces a block of statements in a file. This command is intended to replace a block of statements that are clearly delimited at the beginning and end of the block (such as those added using the `winsblk` command).

**Options**

- `-e "end_string"`
  Specifies a search string that signifies the end of the block of statements. You must surround the string with double quotation marks.

- `-i {"replace_string" | @file_name}
  Specifies a string to replace the text between the delimited statements, or a file containing a block of statements. You must surround the string with double quotation marks.

- `-o output_file`
  Writes the output of this command to the `output_file` file, instead of standard output.

- `-r`
  Replaces the delimiter lines in addition to the block of statements.

- `-s "start_string"
  Specifies a search string that signifies the start of the block of statements. You must surround the string with double quotation marks.

**Examples**

The following example replaces a block of statements beginning with `[boot]` and ending with `end` in the `c:\windows\system.ini` file with statements in the `rplblk.fil` file. The output of this command is redirected to the `c:\temp\output.txt` file.

```
wrplblk -s "[boot]" -e "end" -o c:\temp\output.txt \
   -i @c:\temp\rplblk.fil c:\windows\system.ini
```

**See Also**

`wclrblk`, `winsblk`
wrplline

Replaces a single line in a file. This command should be run from an endpoint.

Syntax

wrplline [-f] -s "search_string" [-o output_file] -r "replace_string" file_name

Description

The wrplline command replaces a line in a text file. The line to be replaced is found using a search string. Output from this command is written to standard output.

Options

- Processes only the first occurrence of the search string. If this option is not specified, the command processes each occurrence of the search string.
- o output_file
  Writes the output of this command to the output_file file, instead of standard output.
- r "replace_string"
  Specifies a string to replace the line that contained the search string. You must surround the string with double quotation marks.
- s "search_string"
  Specifies a search string. If the search string is contained in a line, the line is replaced with the string specified by the -r option. You must surround the string with double quotation marks.

file_name
  Specifies the name of the file in which to replace the line.

Examples

The following example replaces each occurrence of a line beginning with device= in the c:\windows\system.ini file with the type= string. The output of this command is written to the c:\temp\output.txt file.

wrplline -s "device=" -o c:\temp\output.txt -r "type=\n c:\windows\system.ini

See Also

wclrlne, winsline
wrpt

Creates a repeater on a managed node (for both MDist and MDist 2), configures MDist repeaters, and manages MDist distributions.

Syntax

wrpt

wrpt -A [-f] -k distribution_id

wrpt -g [-e]

wrpt -L

wrpt [-n] source_host [always | noalways] [default | nodefault] [wan | nowan] [range=value]

wrpt-q source_host target [target...]

wrpt-R -k distribution_id

wrpt -r source_host

wrpt -T [seconds]

wrpt -t source_host [-k distribution_id] [reinit | keyword=value...]

Description

The wrpt command is used to configure MDist repeaters and manage distributions using the MDist service. It also is used to create repeaters on managed nodes using MDist, MDist 2, or both. For more information about multiplexed distribution services, see Tivoli Management Framework User’s Guide.

The source_host option specifies the name of a source host (managed node). Host numbers are the same as those in the Disp column of the output from the odadmin odlist command.

Note: The wrpt command cannot add endpoints to or delete endpoints from a gateway range. The gateway range is dynamically set as endpoints log in to gateways or are deleted from gateways.

Options

When no options are listed, wrpt returns a table that contains all repeaters. The first column is the source host name with the host number in square brackets ([ ]); the second column indicates that the repeater is a default repeater (d) or a wide area network (WAN) entry site (w); and the third column lists the range of hosts served by the repeater.

Note: The resulting output lists only the managed nodes in a repeater range. Endpoints are not listed.

-A [-f] -k distribution_id

Cancels the specified distribution and ends the active distribution. You
cannot restart a distribution after it has been canceled. The user is prompted with an Are you sure? message unless the force option (–f) is specified. Options are as follows:

–f Forces the cancel operation and suppresses any confirmation prompt.

–k distribution_id
   Specifies the target active process. distribution_id specifies the unique process number of an active distribution. To obtain the value for distribution_id, use the –L option.

Note: Canceling a distribution does not remove files that have already been successfully installed on targets.

–g [–e]
   Changes the format to match the input option format of the wrpt add or change options. This makes it easier to capture a repeater layout to restore later. The –e option lists the range of endpoints served by the repeater.

–L Lists all active distributions in a four-column format. The first column is the unique active distribution number, the second column is the distribution name (a label chosen by the application), the third column is the distribution start time, and the fourth column gives statistics for the distribution in the following format: in/est_size [out_min-out_max].

–n source_host [wan | nowan] [default | nodefault] [always | noalways]
[range=value]
   Creates a new or modifies an existing repeater. Options are as follows:

source_host
   Specifies the source host (managed node) as it is registered in the Tivoli name registry.

always | noalways
   Specifies that distributions go through this repeater although the repeater has only one client. By default, if a repeater has only one client, a distribution to that client goes directly to the client, bypassing the repeater. The always option overrides the default behavior. Use the noalways option to disable the always option and revert to default behavior.

default | nodefault
   Indicates that the repeater serves all hosts that are not explicitly listed in another repeater range. Use nodefault to disable this option.

range=value
   Specifies a comma-separated list of host numbers; consecutive numbers can be abbreviated with a dash (-). For example, 2-14 would be an inclusive list of host numbers 2 through 14. Tivoli Management Framework does not check ranges. If you create conflicting or overlapping ranges, unexpected results can occur.

wan | nowan
   Enables or disables this repeater as the WAN entry point for the region. All distributions from other regions are routed through this repeater. If you do not specify a WAN entry point for the region, any repeater can be the first hop of an interregion distribution.
-q source_host target
Displays the distribution route between the specified source host and target nodes. Output is shown in an indented format, displaying repeaters between the specified source host and targets. Use this option to verify that a route is as you expect before you begin a distribution. Options are as follows:

source_host
Specifies the label, object ID, or name of the source host for the distribution.

target
Specifies the label, object ID, or name of the target for the distribution.

Note: If the managed node and target endpoint have the same name, you can use @ManagedNode and @Endpoint notation.

-r host
Removes the specified repeater. The host option specifies the source host (managed node) as it is registered in the Tivoli name registry.

-t source_host [-k distribution_id] [reinit | keyword=value]
Displays and changes tuning options for a repeater. Options are as follows:

source_host
Specifies the source host (managed node) as it is registered in the Tivoli name registry.

-k distribution_id
Causes configuration options to affect only the active distributions. distribution_id specifies the unique process number of an active distribution. To obtain the value for distribution_id, use the -L option.

reinit
Resets all options to the factory default.

keyword=value
Enables you to specify one of the following keywords and a value (either an integer or an absolute path name for a directory) for the keyword. If value is not specified, the existing options for the specified repeater are displayed. Keywords are as follows:

disk_dir
Specifies the directory used for temporary paging, or swap, space. Note that the repeater must be configured with adequate swap space to avoid hung distributions. This swap space must be at least as large as disk_max.

disk_hiwat
Specifies the disk usage (in KB) at which a delay occurs between disk block allocations. The delay period is specified by the disk_time tuning keyword. A disk block allocation is 16 KB. Tivoli recommends that this value equal about 50 percent of the maximum disk space available to the repeater.

disk_max
Specifies the maximum amount of disk space (in KB) to use for paging. You must set the disk_max and mem_max options dependent on the type of repeater you are distributing through. For nongateway repeaters, mem_max plus disk_max should at least equal the size of the largest
file package distributed if `max_conn` is less than the number of clients that are targets of the distribution. For gateway repeaters, only the `disk_max` option should be set to a value at least equal to the size of the largest file package distributed if `max_conn` is less than the number of clients that are targeted by the distribution.

**disk_time**
Specifies the delay (in seconds) between disk block allocations. The delay starts only after disk usage rises above the number indicated by the `disk_hiwat` tuning keyword.

**max_conn**
Specifies the maximum number of simultaneous parallel client connections initiated by the repeater during a distribution.

**mem_max**
Specifies the maximum memory (in KB) to be used before paging to disk. You must set the `disk_max` and `mem_max` options dependent on the type of repeater you are distributing through. For nongateway repeaters, `mem_max` plus `disk_max` should at least equal the size of the largest file package distributed if `max_conn` is less than the number of clients that are targets of the distribution. For gateway repeaters, only the `disk_max` option should be set to a value at least equal to the size of the largest file package distributed if `max_conn` is less than the number of clients that are targeted by the distribution.

**net_load**
Specifies the maximum amount of data (in KB per second) that the repeater sends to the network for each distribution. Tivoli recommends that this value equal about 25 percent of the network bandwidth (between repeater and clients). Note that you cannot set `net_load` to a value that is greater than 32 MB/sec. Also, if you specify a negative number for `net_load`, the option is set per target machine rather than per distribution.

**net_spacing**
Specifies a delay (in milliseconds) to insert between each write to the network.

**stat_intv**
Specifies a high-level Transmission Control Protocol (TCP) timeout (in seconds) after which an error terminates the blocked connection. This value is dependent on the network client machines processors, particularly the PC processors and RAM.

`–R –k distribution_id`
Returns the distribution route of an active distribution. Use the `–k distribution_id` option to specify the distribution ID of the active distribution. Unlike the `–q` option, it is not necessary to specify the source and destination nodes. To obtain the value for `distribution_id`, use the `–L` option.
Specifies the repeater manager timeout. This timeout value is the maximum time (in seconds) that a repeater node waits after a distribution for final processing on the target to complete before an error is forced to terminate the connection. A final timeout value of 0 indicates no, or infinite, timeout.

### Authorization

#### senior

### Examples

1. To view a list of all repeaters in the region, enter the following command:

```
wrpt
```

```
fuji [1] wd [default]
```

The first column displays the repeater name followed by its dispatcher number in square brackets ([ ]). The first entry in the second column can be a w or a hyphen (-). A w indicates that the entry is a WAN entry site. If the second entry in the second column is a d, the entry is the default repeater for the region. The third column contains the range of hosts served by the repeater. If no range is specified ([ ]), the repeater is the only repeater in the region.

If you specify the `-g` option, the format is changed to match the input option format of the `wrpt` add or change options. This makes it easier to capture a repeater layout to restore later.

2. To cancel a distribution named peppe, enter the following command:

```
wrpt -A peppe
```

3. The following example creates a repeater on the peppe host:

```
wrpt -n peppe range=2-14,18,20-40
```

where `-n peppe` specifies host peppe as a new repeater, and `range=2-14,18,20-40` specifies the host numbers that are in the distribution range for the new repeater. The host numbers include 2 through 14, 18, and 20 through 40.

4. To view the distribution route between a managed node smith and an endpoint jones, enter the following command:

```
wrpt -q smith jones
```

```
--[RPT:smith [1]]
    --[RPT:smth [1]]
    | --[RPT:realty [6]]
    | --jones [5]
```

You also can specify the node label or object ID. In addition, if the managed node and endpoint have the same name, you can use `@ManagedNode` and `@Endpoint` notation. For example, to view the distribution route between a managed node and an endpoint smith, enter the following command:

```
wrpt -q @ManagedNode:smith @Endpoint:smith
```

```
--[RPT:smith [1]]
    --[RPT:Jones [3]]
    | --smith [4]
```
5. To list all active distributions, enter the following command from the source repeater machine:

```
wrpt -L
```

```
4 fp_distribute 05 09 16:12:50 2816/0 [640-640]
```

where 4 is the unique active distribution number, fp_distribute is the distribution name (a label assigned by the application), 05 09 16:12:50 is the distribution start date and time, and 2816/0 [640-640] provides statistics for the distribution.

Statistics indicated in the last column are displayed in the following format: in/est_size [out_min-out_max] where in indicates the amount of data the repeater received, est_size indicates the estimated size of the distribution, out_min indicates the number of bytes sent to the slowest target, and out_max indicates the number of bytes sent to the fastest target.

6. To view settings for the peppe repeater, enter the following command:

```
wrpt -t peppe
```

```
mem_max = 10000
disk_max = 50000
disk_hiwat = 50000
disk_time = 1
disk_dir = "C:/TEMP/"
net_load = 500
max_conn = 100
stat_intv = 180
```

7. The following example gets the unique process number for an active distribution, calls that distribution using its number, and changes the maximum network load that the process can add to 100 kilobytes per second. This command changes the network load only for this distribution.

```
wrpt -L
```

```
1 fp_distribute Jun 16 12:53:27 1696/1696 [0-432]
```

```
wrpt -k 1 -t peppe net_load=100
```

**Note:** peppe is the name of the managed node that the repeater is configured on.
**wrunas**

Runs a given command as a given user. The password for the user is retrieved from the registry using a given key.

**Syntax**

```
wrunas [user_name | key | command]
```

**Description**

The `wrunas` command retrieves the password from the registry, uses the Microsoft authentication package, and launches a given command. The `wrunas` command can be used in a task library script or a Tivoli Application Extension Facility (AEF) method script to launch executables from the Tivoli desktop. These scripts must be installed to run as $root_user.

**Options**

- **command**
  Specifies the command to be executed.
- **key**
  Specifies the key that stores the password for the user name.
- **user_name**
  Specifies the user name.

**Examples**

The following example retrieves the password for Administrator when the key is `admin_key`:

```
wrunas Administrator admin_key net config workstation
```
wruninvquery

Queries the database for inventory information and returns a list of object IDs and object labels of machines that match the query criteria.

Syntax

```
 wruninvquery [-i] [-T idl_type] [-I | -t] query_name [input...]
```

Description

The `wruninvquery` command runs a query and returns a list of object IDs and object labels in a format that can be used for a subscription list. The output of all queries is a `SysAdminTypes_ObjectLabelList` type. To use this command, the columns `TME_OBJECT_ID` and `TME_OBJECT_LABEL` must be included in the columns list of the query. If these columns are not included in the query, or if you want to simply display query output in text format, use the `wrunquery` command.

The `wruninvquery` command returns only the following output:
- A line-separated list of object IDs (the default)
- A line-separated list of object labels
- An American National Standard Code for Information Interchange (ASCII) Interface Definition Language (IDL)-encoded version of an instance of the `SysAdminTypes_ObjectLabelList` type

This command can also read a limited set of inputs that can narrow the query results. If input is provided, the query combines the results of the query with the input, and then returns only the results that are found in both lists. Input types are accepted in the following formats:
- A list of object IDs (the default) separated by spaces.
- An ASCII representation of an IDL type. Currently, the only IDL types that are valid input are the following:
  - `SysAdminTypes_ObjectList`
  - `SysAdminTypes_ObjectLabelList`
  - `TMF_CCMS_subscriber_list`

Options

- **-i** Reads query input, either object IDs or an ASCII representation of the IDL data type, from standard input. If the `-T` option is not specified, the input is interpreted as a space-separated list of object IDs. If the `-T` option is specified, the input must be an ASCII representation of the IDL data type (similar to that used by the `idlcall` and `idlattr` commands). Input can come from either standard input or the command line, but not both.

- **-l** Specifies that the output should be a new line-separated list of object labels.

- **-t** Specifies that the output should be an ASCII representation of the IDL result as output.

- **-T idl_type** Specifies the full name of an IDL data type. The input, whether coming from standard input or the command line, must be an ASCII representation of the IDL type.
wruninvquery

**input** Specifies the input to the query. If the –T option is not specified, the input is interpreted as a space-separated list of object IDs. If the –T option is specified, the input must be an ASCII representation of the IDL data type (similar to that used by the `idlcall` and `idlattr` commands). Input can come from either standard input or the command line, but not both.

**query_name** Specifies the name of the query to run.

**Authorization**

query_execute, admin, senior, or super

**Examples**

1. The following example runs the AIX-machines query and prints the output as a list of object IDs:
   `wruninvquery AIX-machines`
   ```
   1922582407.1.323#TMF_ManagedNode::Managed_Node#
   555555.1.332#TMF_ManagedNode::Managed_Node#
   ```
2. The following example runs the AIX-machines query and prints the output as a list of object labels:
   `wruninvquery -l AIX-machines`
   ```
   manzano
   amon-sul
   ```
3. The following example runs the AIX-machines query and prints the output as an ASCII-encoded representation of the SysAdminTypes_ObjectLabelList type:
   `wruninvquery -t AIX-machines`
   ```
   { 2 } 1922582407.1.323#TMF_ManagedNode::Managed_Node# "manzano"{ } 555555.1.332#TMF_ManagedNode::Managed_Node# "amon-sul"{ }
   ```
4. The following example runs the AIX-machines query, using the subscribers of the pm1 profile manager, and produces a list of labels for the output:
   ```
   idlcall 555555.1.535#TMF_CCMS::ProfileManager# \
   _get_subscribers | wruninvquery -l -i \
   -T TMF_CCMS::subscriber_list aix-boxes amon-sul
   ```
   ```
   manzano
   ```

**See Also**

[idlattr] [idlcall] [wcrtqlib] [wcrtquery] [wgetquery] [wrunquery] [wsetquery]
**wrunjob**

Runs a job in a task library.

**Syntax**

```
wrunjob job_name -l library [-a option] [-e name=value] [-i] [-E] [-r] [-T transaction_type]
```

**Description**

The `wrunjob` command runs a job that exists in a task library.

**Options**

- `-a option`  
  Specifies the argument to be passed to the task. If the argument to be passed includes an option flag and an argument, enclose both in quotation marks (for example, `-a "-o option"`).

- `-e name=value`  
  Sets an environment variable for the task (for example, `DISPLAY=bald:0.0`).

- `-E`  
  Passes all user environment variables to the task.

- `-i`  
  Reads input options for the task from standard input.

- `-l library`  
  Specifies the task library containing the task to be run.

- `-r`  
  Returns an error code of 1 if at least one endpoint fails to execute its job properly.

- `-T transaction_type`  
  Specifies a transaction type. This option can be one of the following:

  - `none`  
    No transaction.

  - `revoke`  
    Revocable transaction.

  - `sub`  
    Subtransaction.

  - `top`  
    Top-level transaction. This is the default if the `-T` option is not specified.

- `job_name`  
  Specifies the name of the job to be run.

**Authorization**

You must have the role specified when the job was created. Use the `wgetjob` command to find the required role.

**Examples**

1. The following example runs the `date_job` job from the `my_tasks` task library:
   
   ```
   wrunjob date_job -l my_tasks
   
   ########################################################################
   Task Name: date_task
   Managed Node: bald
   Return Code: 0
   -------Standard Output-------
   ```
2. The following example runs the date_job job from the my_tasks task library and uses the LANG variable set to German:

\[\text{wrunjob date_job -l my_tasks -e LANG=de}\]

Task Name: date_task
Managed Node: bald
Return Code: 0

3. The following example runs the ps_vernon job from the NoonTide task library and passes the aux option:

\[\text{wrunjob ps_vernon -l NoonTide -a aux}\]

Task Name: ps
Task Endpoint: Vernon (ManagedNode)
Return Code: 0

See Also

**wrunjob** | **wcrtjob** | **wcrttask** | **wgetjob**
wrunquery

Runs a query and returns the results to either standard output or a file.

**Syntax**

```
wrunquery [-n] [[-h host_name] -f file_name] [-d delimiter] query_name
```

**Description**

The `wrunquery` command runs a query and enables you to either display the results or save them to a file. By default, the `wrunquery` command returns the output to standard output. To get a list of object IDs and object labels to use for a subscription list, use the `wruninvquery` command.

**Options**

`-d delimiter`

Specifies the delimiter that separates the entries in the output file. The default delimiter is a comma.

`-f file_name`

Specifies the path and name of the file in which to store the query results.

`-h host_name`

Specifies the name of the managed node on which to store the query results. If you do not specify a managed node, the file is saved on the local machine.

`-n`

Omits the headers from the output.

`query_name`

Specifies the name of the query to run.

**Authorization**

`query_execute`, `RIM_view`, `admin`, `senior`, or `super`

**Examples**

The following command runs the Operating-systems query and sends the output to the query.txt file on managed node amon-sul. The output file contains headers. The entries are separated by semicolons.

```
wrunquery -h amon-sul -f query.txt -d ";" Operating-systems
```

The query.txt file contains the following:

```
Query Name: Operating-systems
Number of rows: 9
BOOTED_OS_VERSION_TYPE BOOTED_OS_NAME PROCESSOR_SPEED
3.2;AIX;UNKNOWN
3.2;AIX;UNKNOWN
3.2;AIX;UNKNOWN
4.1;AIX;UNKNOWN
4.1;AIX;UNKNOWN
3.10;Windows 25;486 DX
3.10;Windows 25;486 DX
3.10;Windows 33;486 DX
3.10;Windows 133;Intel Pentium
```
wrunquery

See Also

wcrqlib, wcrquery, wgetquery, wruninvquery, wsetquery
**wruntask**

Runs a task in a task library.

**Syntax**

```
```

**Description**

The `wruntask` command runs a task in a task library. The task must be previously created.

**Options**

- **-a option**
  
  Specifies the argument to be passed to the task. If the argument to be passed includes an option flag and an argument, enclose both in quotation marks. For example:
  
  `-a "-o option"`

  The `-a` option must be repeated for each option specified. For example:
  
  `-a "-h node" -a "-p profile_mgr"`

- **-e name=value**
  
  Sets an environment variable for the task. The `-e` option must be repeated for each environment variable specified. For example:
  
  `-e DISPLAY=bald:0.0 -e COLOR=red`

- **-E**
  
  Passes to the task all environment variables set in your current shell.

- **-h node...**
  
  Specifies the nodes (managed nodes and endpoints) on which to run the task. You must specify at least one node or at least one profile manager (with the `-p` option). The `-h` option must be repeated for each node specified. For example:
  
  `-h vernon -h everest -h fuji`

- **-i**
  
  Reads standard input and passes it to the task as its standard input.

- **-l library_name**
  
  Specifies the task library containing the task to be run.

- **-m timeout**
  
  Specifies the amount of time in seconds that the task library waits for results to be returned from the task. This option does not affect the execution of the task on managed nodes. If you do not use the `-m` option, the default timeout is 60 seconds.

  **Note:** When executing tasks against endpoints, the task timeout value must be greater than the gateway timeout value. For example, if the task timeout value is 600 and the gateway timeout is 300, the task fails after 300 seconds.

- **-M mode**
  
  Specifies the mode in which the task runs. Valid options are the following:
**parallel**
Runs the task on all specified managed nodes and any subscribers simultaneously. This is the default if you do not specify the `−M` option.

**serial**  Runs the task on one managed node or endpoint at a time.

**staged** Runs the task on a set number of managed nodes and endpoints at specified intervals. If you specify this mode, you must use the `−n` and `−s` options.

`−n number`
Specifies the number of nodes (managed nodes or endpoints) on which to run the task in each stage. You must supply a value for this option if you specified the `-M staged` option. The maximum number of nodes is 150.

`−o output_format`
Defines the format of the task output. Task execution output format is specified with an octal number from 0 to 17. The format is constructed by adding the value of the desired output. For example, to print the return code to standard output, specify `−o 05`. To output to a file, use the standard redirection syntax. Output values are as follows:

- `01`: Prints a descriptive header for each record
- `02`: Prints the return code
- `04`: Prints the standard output
- `08`: Prints the standard error output

`−p profile_mgr...`
Specifies the profile managers on which the task runs. You must specify at least one profile manager or at least one node (with the `−h` option). The `−p` option must be repeated for each profile manager specified. For example:
   `-p pm1 -p pm2 -p pm3`

`−r` Returns an error code of 1 if at least one endpoint fails to execute its job properly.

`−s interval`
Specifies the number of seconds between when the task runs on one group of managed nodes and when it runs on the next group. The interval is at the end of the first group. You must supply a value for this option if you specified the `-M staged` option.

`−t task_name`
Specifies the name of the task to be run.

`−T trans_type`
Specifies a transaction type. See “Tivoli transactions” on page 6 for additional information about transaction types. This option can be one of the following:

- `none`: No transaction.
- `revoke`: Revocable transaction.
- `sub`: Subtransaction.
- `top`: Top-level transaction. This is the default if the `-T` option is not specified.
Authorization

You must have the role that was specified when the task was created. You must have this role in the region to which the target for the task belongs. You can use the `wgettask` command to find the required role.

Examples

1. The following example runs the `date_task` task on nodes bald and fuji. The task is contained in the `my_tasks` task library.

   `wruntask -t date_task -l my_tasks -h bald -h fuji`

   Task Name: `date_task`
   Managed Node: `bald`
   Return Code: `0`
   -------Standard Output-------
   Mon Nov 21 10:49:34 CST 1998
   -------Standard Error Output-------

2. The following example runs the `date_task2` task on node bald. The task is contained in the `my_tasks` task library. The output of this task is standard output.

   `wruntask -t date_task2 -l my_tasks -h bald -o 04`

   Task Name: `date_task`
   Managed Node: `bald`
   Return Code: `0`
   -------Standard Output-------
   Mon Nov 21 10:49:45 CST 1998

3. The following example runs the `ps` task on node vernon. The task resides in the NoonTide task library. The example passes the `aux` option to the task.

   `wruntask -t ps -l NoonTide -h vernon -a aux`

   Task Name: `ps`
   Task Endpoint: `vernon` (ManagedNode)
   Return Code: `0`
   -------Standard Output-------
   USER     PID  %CPU %MEM    SZ   RSS  TT  STAT  START      TIME COMMAND
   root     2245  54.5   6.2  360 2368  ?   S  13:10  0:02 task_endpoint
   root     2244  19.7  1.3  228  488  ?   S  13:10  0:00 /tmp/ taskAAAa02245 aux
   nobody   2243  14.6  5.2  176 1992  ?   S  13:10  0:01 man_node_skel1
   nobody   2239  12.2  7.3  568 2800  ?   S  13:10  0:02 repository_skel1
   root     134   8.4  2.9  1508 1100  ?   S  13:09  0:01 oserv -p 94 \ -k /usr/Tivo
   nobody   2237  3.5  5.7  184 2188  ?   S  13:10  0:01 library_skel1
   root     2236  1.1  5.5  120 2108  p0  S  13:09  0:01 wrunjjob ps_ vernon -l Noo
   root     172   0.0  1.8  136  684  ?   S  13:09  0:02 ./usr/lndk

See Also

- `wcrtjob`
- `wcrtask`
- `wgettask`
wschedjob

Schedules a job that exists in a task library.

**Syntax**

```plaintext
wschedjob -n name -L library_name -t "mm/dd/yyyy hh:mm" [-c 'time_period'] [-C daytime | nighttime | weekday | weekend from to] [-D] [-d desktop] [-f file -h host] [-g group] [-l label] [-m email] [-o] [-r 'time_period' | 'iterations'] [-R 'time_period' | 'iterations'] [-s description]
```

**Description**

The `wschedjob` command allows administrators to schedule a job. Only jobs that exist in a task library can be scheduled from the command line.

Administrators must have the proper authorization to run the job. If the administrator does not have the proper authorization at the time the job is run, it fails. Before scheduling a job, administrators should verify that they have the proper authorization by running the job manually, for example by using the `wrunjob` command.

**Options**

- `-c 'time_period'`
  Specifies when a job is canceled if it did not start as scheduled. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, or day. For example, if you specify '3 hour', the job is canceled 3 hours after its originally scheduled start time.

- `-C daytime | nighttime | weekday | weekend from to`
  Specifies the conditions or restrictions under which the job runs. The `from` option can be either a starting day or a starting time. The `to` option can be either an ending day or an ending time. Times must be entered using a 24-hour clock (for example, 9:00 for 9 a.m. or 14:00 for 2 p.m.). Days must be entered as numeric values where Sunday is 0 and Saturday is 6. Valid options are as follows:

  - `'daytime from to'`
    Specifies that the job runs only during the day between the hours of `from` and `to`.

  - `'nighttime from to'`
    Specifies that the job runs only during the night between the hours of `from` and `to`.

  - `'weekday from to'`
    Specifies that the job runs only during weekdays on and between the days indicated by `from` and `to`.

  - `'weekend from to'`
    Specifies that the job runs only during the weekend on and between the days indicated by `from` and `to`.

- `-d desktop`
  Specifies which desktop displays the Status window when any action is performed on the job. Multiple desktops can be specified.

- `-D` Disables the job. The job remains in the scheduler but does not run until it is enabled.
-f file  Specifies the file to which the job status is written when any action is performed on the job. If a file is specified, the –h option must be used to specify a host.

-g group  Specifies the notice group to which the job status is sent when any action is performed on the job. Multiple notice groups can be specified.

-h host  Specifies the host on which the job status file is to be written. Must be used with the –f option.

-l label  Specifies the name specific to this instance of the job.

-L library_name  Specifies the name of the task library where the job resides. This option is required.

-m email  Specifies the e-mail address to which the job status is sent when any action is performed on the job. Multiple e-mail addresses can be specified.

-n name  Specifies the name of the job with the task library that is scheduled. This option is required.

-o  Specifies that the time indicated with the –t option is in the past. Overrides warning message.

-r ’time_period’ | ’iterations’  Specifies the repeat information. If the iterations option is specified, the job repeats for a finite number of times. The unit of time must be specified in the local language.

 ’time_period’  Specifies how often a job is retried. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, day, week, month, or year. For example, if you specify ’3 hour’, the job is repeated every 3 hours.

 ’iterations’  Specifies how many times a job is repeated. You must specify an amount of time, a unit of time, and a number of times. The unit of time must be minute, hour, day, week, month, or year. For example, if you specify ’3 hour 6’, the job is repeated every 3 hours until it has been repeated six times.

-R ’time_period’ | ’iterations’  Specifies the retry information. If the iterations option is specified, the job retries for a finite number of times.

 ’time_period’  Specifies how often a job is repeated. You must specify a number (amount of time) and a unit of time. The unit of time must be minute, hour, or day. For example, if you specify ’3 hour’, the job is retried every 3 hours.

 ’iterations’  Specifies how many times a job is retried. You must specify an amount of time, a unit of time, and a number of times. The unit of
wschedjob

time must be minute, hour, or day. For example, if you specify '3 hour 6', the job is retried every 3 hours until it has been tried six times.

-s description
    Describes the job. Multiword descriptions require quotation marks.

-t "mm/dd/yyyy hh:mm"
    Specifies the time the job is scheduled to initially execute. This option is required.

Authorization
    admin for the Scheduler resource

Examples

1. The following example schedules the job SendWishList from the task library, Holiday. The job executes at 6:00 a.m. on December 24, 1998. When any action is performed on the job, a job status message is sent to the e-mail address santa@npole.com.
   wschedjob -t "12/24/1998 6:00" -m santa@npole.com -L \ Holiday -n SendWishList

2. The following example schedules the job SendWishList from the task library, Holiday. The job executes at 6:00 a.m. on December 24, 1998. In this example, when an action is performed on the job, a job status message is sent to e-mail addresses santa@npole.com, mom@home.com, and dad@home.com. The job repeats every 5 minutes. If the job fails, it retries once, 1 minute after failure.
   wschedjob -t "12/24/1998 6:00" -m santa@npole.com \ -L Holiday -n SendWishList -r '5 minute' \ -m mom@home.com -m dad@home.com -R '1 minute 1'

3. The following example schedules the job nice_list from the task library, MakeToys. The job runs Monday through Friday at 10 p.m.
   wschedjob -t "3/4/1996 22:00" -L MakeToys -n nice_list \ -r '1 day' -C 'weekday 1 5'

See Also
    wdelsched|wedsched|wenblsched|wgtsched
wserv=

Installs the Tivoli management region server on UNIX machines.

Syntax

```bash
wserv -c cdrom_path [-a server_name] [-d] [-P] [-p path_prefix[]]
[install_variable=value...]
```

Description

The `wserv` command installs the initial the Tivoli server for a Tivoli region. Two modes of installation are supported: X11-based installation and command-line only installation. To use the X11 installation, make sure that the DISPLAY environment variable is set and that the DOGUI environment variable is not set. To use the command-line only installation, make sure that the DOGUI environment variable is set to no. By default, the X11 version of installation is chosen. In either case, change directory to `installation_directory`; that is, the directory where you ran the `wpreinst.sh` script or to the directory where you un-tarred the file0.tar file. Alternatively, you could set the BINDIR environment variable `toinstall_directory`.

Options

- `-a server_name`
  Specifies the name of the Tivoli server. You can specify either the local host name or a remote host name. The default is the local host name. On systems with host names longer than eight characters, the `hostname` command returns `unknown`. This option lets you fix that behavior.

  **Note:** If you are installing on the local host and specifying a fully qualified host name, the appropriate remote access must be enabled by updating the local `.rhosts` file.

- `-c cdrom_path`
  Specifies the path to the CD-ROM image.

- `-d`
  Sets the installation variable (`install_variable`) to its default value. This flag is used for only the command-line only version. This option is required unless you select between each and every one of the options and environment variables.

- `-p path_prefix[]`
  Attaches the specified path prefix to the beginning of the default installation paths. If the optional exclamation mark (!) is present, `path_prefix` is prepended to only the last component of the default installation paths. For example, the default installation path for the binaries is `/usr/local/Tivoli/bin`. If you specify `-p/Tivoli`, the installation path would be `/Tivoli/usr/local/Tivoli/bin`. If you specify `-p/Tivoli!`, the path would be `/Tivoli/bin`.

- `-P`
  Specifies the use of a global root password instead of trusted host access. This option is used only when installing on a remote host.

  `install_variable=value`
  Specifies variables that control the installation. The variables can be set or the defaulted used on the command line. If you are using the X11 version, you can change these values during the installation. When installing from the command line, the only way to set these values is to pass them on the
command line. Use the installation variables to specify the required information or to override default information.

**Note:** Type the names of the installation options exactly as denoted in the documentation. Installation options are case sensitive.

Several of the installation variables specify the directories where the Tivoli server is installed. If a directory already contains files from a previous installation, the command does not recopy the files. You can force any of these directories to be reinstalled by entering the `!` character after the specified directory. The following are the installation variables related to directories:

- **BIN**=`binaries_dir`
  Overrides the default installation path (`/usr/local/Tivoli/bin`) for Tivoli Management Framework binaries.

- **LIB**=`libraries_dir`
  Overrides the default installation path (`/usr/local/Tivoli/lib`) for Tivoli Management Framework libraries.

- **ALIDB**=`server_database_dir`
  Overrides the default installation path (`/var/spool/Tivoli`) for Tivoli Management Framework server database.

- **MAN**=`manpage_dir`
  Overrides the default installation path (`/usr/local/Tivoli/man`) for Tivoli Management Framework manual pages.

- **APPD**=`X11_dir`
  Overrides the default installation path (`/usr/lib/X11/app-defaults`) for the X11 application defaults.

- **CAT**=`message_catalog_dir`
  Overrides the default installation path (`/usr/local/Tivoli/msg_cat`) for Tivoli Management Framework message catalogs.

The following are other useful installation variables:

- **@EL@**=`None | Simple | DES`
  Defines the encryption level to be used when installing the server. The default level is **Simple**.

- **@ForceBind@**=`yes | no`
  Forces communication connections to bind to a single Internet Protocol (IP) address. This option is used in certain high availability or failover configurations where multiple object dispatchers reside at different IP addresses on a single physical system.

- **AutoStart**=`0 | 1`
  Indicates whether the Tivoli daemon should be started at system boot time. By default, the daemon is not started.

- **CreatePaths**=`0 | 1`
  Indicates whether to create (1) the specified directories if they do not already exist. By default, the directories are created. It is considered an error if a directory that you specified with `install_variable` does not exist.
IP=install_password
Sets the installation password. By default, there is no password. This password is the installation password when you install Tivoli Management Framework on clients, the default seed when you are using encryption, and the interregion password when you are connecting Tivoli regions using encryption.

LK=license_key
Specifies the license key. This value specified for this installation variable is ignored.

RN=region_name
Overrides the default policy region name. The default policy region name can be changed later.

SetPort=0 | 1
Indicates whether to configure the remote start capability of the Tivoli daemon. By default, this capability is not configured.

Authorization
Root access on the system being installed.

Files
/tmp/tivoli.sinstall
Contains verbose debugging information from the latest installation attempt.

/tmp/install.cfg.error /tmp/install.cfg.output
Transient file created during the initialization of the Tivoli management region server database. After initialization, if there are no errors, these files are removed.

/etc/Tivoli/setup_env.sh
A file that can be sourced in from Bourne shell compatible shells after installation that contains useful shell environment variables.

/etc/Tivoli/setup_env.csh
A file that can be sourced in from C shell compatible shells after installation that contains useful shell environment variables.

Environment Variables
A number of environment variables have installation implications:

DISPLAY
Specifies the X11 display to be used for installation.

DOGUI
If set to something other than the value of the DISPLAY variable, the command line version is used.

EtcTivoli
The default is the /etc/Tivoli directory. Do not override.

BINDIR
If you do not want to run this command from the install_dir directory (the directory where you ran the wpreinst.sh command or the directory where you untarred the file0.tar file) you can set this variable to your install_dir directory.
o_dispatch

The default port is 94. Do not override.

SAVE_CFG_FILES

The debugging files used during initialization of the Tivoli server, /tmp/install.cfg.output and /tmp/install.cfg.error, are removed after a successful installation. If you want to keep them, you can set this variable to a nonnull value.

Note: When you install Tivoli Management Framework on a UNIX operating system, the /tmp/.tivoli directory is created. This directory contains files that are required by the object dispatcher process. You should not delete this directory or any of its contents unless explicitly directed to by your Tivoli support provider. You should also ensure that regularly scheduled disk clean-up jobs (cron or Tivoli jobs) do not remove this directory or its contents.

To use a different directory, you must set an environment variable in both the object dispatcher and the shell. After installing Tivoli Management Framework, perform the following steps to set the necessary environment variables:

1. Create a directory. This directory must have at least public read and write permissions. However, define full permissions and set the sticky bit to ensure that users cannot modify files that they do not own.

2. Set the environment variable in the object dispatcher:
   a. Enter the following command:
      
      odadmin environ get > envfile
   b. Add the following line to the envfile file and save it:
      
      TIVOLI_COMM_DIR=new_directory_name
   c. Enter the following command:
      
      odadmin environ set < envfile

3. Edit the Tivoli-provided setup_env.csh, setup_env.sh, and oserv.rc files in the /etc/Tivoli directory to set the TIVOLI_COMM_DIR variable.

4. For HP-UX and Solaris systems, add the following line to the file that starts the object dispatcher:

   TIVOLI_COMM_DIR=new_directory_name

   Insert the line near where the other environment variables are set, in a location that runs before the object dispatcher is started. The following list contains the file that needs to be changed on each operating system:
   • For HP-UX operating systems: /sbin/init.d/Tivoli
   • For Solaris operating environments: /etc/rc3.d/S99Tivoli

5. Shut down the object dispatcher by entering the following command:

   odadmin shutdown all

6. Restart the object dispatcher on the Tivoli server by entering the following command:

   odadmin reexec 1

7. Restart the object dispatcher on the managed nodes by entering the following command:

   odadmin reexec clients
Examples

The following examples show command line installations. The DOGUI environment variable is set to no. The X11 version of installation is similar.

1. The following example installs the Tivoli server on the local machine. The complete path to the CD-ROM image is /cdrom. The binaries are installed in /Tivoli/bin. The libraries are installed in /Tivoli/lib. The server database is installed in /Tivoli/database. The manpages are installed in /Tivoli/man. The X11 defaults are installed in /Tivoli/X11. The message catalogs are installed in /Tivoli/cat. A default policy region is created with the name NoonTide-Region. The Tivoli daemon automatically starts at system boot. The remote start capability of the Tivoli daemon is configured. The specified directories are created if they do not exist. The installation password is set to Tivoli4Ever. The default encryption level is not used.

    ./wserver -c /cdrom/cdrom0 BIN=/Tivoli/bin \ 
    LIB=/Tivoli/lib ALIDB=/Tivoli/database MAN=/Tivoli/man \ 
    APPD=/Tivoli/X11 CAT=/Tivoli/cat RN=NoonTide-Region \ 
    AutoStart=1 SetPort=1 CreatePaths=1 IP=Tivoli4Ever

Note: To reinstall this Tivoli server, force the installation to overwrite the directory containing the Tivoli database (ALIDB=!). You can optionally overwrite any other directories. Use the ! character to force the installation to overwrite directories that already exist. The following example reinstalls the Tivoli server installed with the preceding command, overwriting each directory:

    ./wserver -c /cdrom/cdrom0 \ 
    BIN=! LIB=! ALIDB=! MAN=! APPD=! CAT=! \ 
    RN=NoonTide-Region AutoStart=1 \ 
    SetPort=1 CreatePaths=1 IP=Tivoli4Ever

2. The following example installs the Tivoli server on the local machine. The complete path to the CD-ROM image is /cdrom. The binaries are installed in /Tivoli/bin. The libraries are installed in /Tivoli/lib. The server database is installed in /Tivoli/database. The manpages are installed in /Tivoli/man. A default policy region is created with the name NoonTide.

    wserver -c /cdrom -d -p /Tivoli! ALIDB=/database \ 
    RN=NoonTide

3. The following example installs the Tivoli server on the local machine. The complete path to the CD-ROM image is /cdrom. The binaries are installed in /Tivoli/bin. The libraries are installed in /Tivoli/lib. The server database is installed in /Tivoli/database. The manpages are installed in /Tivoli/man. A default policy region is created with the name NoonTide.

    wserver -c /cdrom BIN=/Tivoli/bin LIB=/Tivoli/lib \ 
    ALIDB=/Tivoli/database MAN=/Tivoli/man RN=NoonTide

4. The following example installs the Tivoli server on the local machine. The complete path to the CD-ROM image is /cdrom. Everything (binaries, libraries, and so on) is installed in the default locations under the user-specified directory /Tivoli.

    wserver -c /cdrom -d

5. The following example installs the Tivoli server on cook, a remote machine. The complete path to the CD-ROM image is /cdrom, and this path must be reachable from both the local machine and the remote machine. The user is prompted to enter the root password on cook. Everything (binaries, libraries, and so on) is installed in the default locations.

    wserver -c /cdrom -P -d -a cook
See Also

wclient
winstall
wsetadmin

Changes information about a Tivoli administrator.

Syntax

```
```

Description

The `wsetadmin` command changes select properties of an existing Tivoli administrator. You can use this command to add or remove logins, to add or remove notice group subscriptions, and change authorization group roles.

**Note:** You cannot use this command to change the user name, the group name, or the label of the Administrator icon. You must make these changes from the Tivoli desktop.

Options

- `-l login`
  Adds the specified login.
- `-L login`
  Removes the specified login.
- `-n notice_group`
  Adds a subscription to a notice group.
- `-N notice_group`
  Removes a subscription from a notice group.
- `-r group,role:role`
  Changes the administrator role in the specified group to the specified role. The following are examples of valid formats for `group`:
  - `@Administration`
  - `@PolicyRegion:Administration`
  - `/Regions/PolicyRegion:Administration`
- `-R group`
  Removes all roles for the administrator in the group.

`name` Specifies the name of the administrator whose properties to change.

Authorization

`senior` or `super`

Examples

1. The following example changes information for administrator Steve Callahan. The administrator role in the Accounting policy region is `admin`. He is no longer subscribed to the Tivoli Authorization notice group, and the login `callahan@teton` is added.

```
wsetadmin -r @Accounting,admin -N "Tivoli Authorization" \   -l callahan@teton "Steve Callahan"
```
2. The following example again changes information for administrator Steve Callahan. The example removes his authorization in the Accounting policy region. It adds a subscription to the Tivoli Authorization notice group, and removes the login callahan@teton.

```bash
wsetadmin -R @Accounting -n "Tivoli Authorization" \
-L callahan@teton "Steve Callahan"
```

See Also

- wcrtadmin
- wgetadmin
wsetdfpol

Sets the default policy objects for a managed object class.

Syntax

wsetdfpol -d class label
wsetdfpol -v class label

Description

The wsetdfpol command sets the default policy objects for a managed node class. The default policy objects are the default policy object and the validation policy object. These policy objects are for the current policy region. When a managed object class is added to a policy region, it receives the specified default policy objects: a policy default object generates default attribute values for resources created in the policy region. A policy validation object validates attribute values for a managed object class.

Options

- `–d` Sets the default policy for the default object.
- `–v` Sets the default policy for the validation object.
- `class` Specifies the class to be set for the default policy object.
- `label` Specifies the label of the desired policy object.

Authorization

admin, senior, super

Examples

The following example makes the policy validation object, Restricted, the default policy validation object for the ProfileManager class.

wsetdfpol -v ProfileManager Restricted

See Also

wgetdfpol
wseterr

Sets the return code from a batch file for a configuration program. This command should be run from an endpoint.

Syntax

```
wseterr return_code
```

Description

The `wseterr` command sets the return code for a batch file invoked as a configuration program. You should use this command at the end of all batch files to return the proper code to the calling Tivoli Enterprise product.

Options

```
return_code
```

Specifies the return code to be returned.
wsetjob

Sets the properties of a job.

Syntax

```bash
```

Description

The `wsetjob` command sets the properties of a job using the specified task.

Options

- `–d node_name`
  Specifies the managed node on which to save the job output.
- `–D`
  Displays the job output to the desktop.
- `–f file_name`
  Specifies the file name in which to save the job output.
- `–h node_name`
  Specifies the managed node on which to run the job.
- `–j job_name`
  Specifies the name of the job being created.
- `–l library_name`
  Specifies the task library containing the task to be included in the job.
- `–m timeout`
  Specifies the number of seconds the task library waits for results to be returned from the task. If you are using `staged` mode, the timeout must be smaller than the interval time.
- `–M mode`
  Specifies the mode in which the job runs. Valid options are the following:
    - `parallel`
      Runs the job on one managed node at a time.
    - `serial`
      Runs the job on all specified managed nodes and any subscribers simultaneously.
    - `staged`
      Runs the job in groups of managed nodes at specified intervals. When you specify `staged` mode, you must also specify the `–s`, `–n`, and `–m` options.
- `–n number`
  Specifies the number of managed nodes in each group in `staged` mode. You must specify a value for this option if you selected `staged` mode.
- `–N`
  Disables writing job output to the file.
- `–o output_format`
  Defines the format of the job output. The job output contains a summary of the job on each managed node. Job output format is specified with a number from 0 to 15. The format is constructed from the sum of any of the following values:
wsetjob

1  Prints a descriptive header for each record.
2  Prints the job return code.
4  Prints the standard output.
8  Prints the standard error output.

-p profile_manager_name
   Specifies the profile manager on which the job runs.

-s interval
   Specifies the number of seconds between when the task runs on one group
   of managed nodes and when it runs on the next group. You must specify a
   value for this option if you selected staged mode. The interval must be
   greater than the timeout value given with the -m option.

-t task_name
   Specifies the name of the task to include in the task library.

-X
   Disables writing job output to the desktop.

Authorization
   admin, senior, super

See Also
   wcrtask, wdeljob, wrunjob, wsettask
wsetlang

Sets the locale for Tivoli method execution on a Tivoli server or managed node.

Syntax

wsetlang [-o] [-l locale_name]

Description

The wsetlang command sets the language environment for a Tivoli server or managed node. The specified locale name can either be a valid operating system locale name or a standard locale name. A locale name is composed of the two-letter ISO 639 language code followed by an optional underscore (_) and two-letter ISO 3166 country code. The standard syntax is as follows:

ll[TT]

where ll represents the language code and TT is the optional country code.

The operating system locale names on UNIX machines can be listed using the following UNIX command:

locale -a

On UNIX operating systems, the specified locale name is mapped to the installed operating system locale name that it most closely matches. If no matching locale name is found, the C locale is used. For Windows operating systems use the specified locale name. It is not validated.

Standard locale names include the following:

- **en** or **C**
  - English
- **fr**
  - French
- **de**
  - German
- **it**
  - Italian
- **ja**
  - Japanese
- **pt_BR**
  - Brazilian Portuguese
- **ko**
  - Korean
- **zh_CN**
  - Simplified Chinese
- **zh_TW**
  - Traditional Chinese

Options

- **-l locale_name**
  - Specifies the desired locale. If the -l option is not specified, the current language environment is used.
- **-o**
  - Sets the language for this Tivoli server or managed node to the specified value.
Authorization

super or senior

Examples

1. To set the locale to French and update the odadmin environment for methods, enter the following command:
   `wsetlang -o -l fr`

   The method environment settings can be viewed using the following command:
   `odadmin environ get`

2. To just view the operating system locale name for French, enter the following command. This does not update the method environment.
   `wsetlang -l fr`
**wsetpkey**

Encrypts and stores a password in the registry.

**Syntax**

```bash
wsetpkey -a key [-k]
```

```bash
wsetpkey -d key
```

**Description**

The `wsetpkey` command encrypts and stores a password in the Windows registry. The encrypted password is stored under a key you provide.

**Options**

- `-a key` Adds or replaces a key and password.
- `-d key` Reads the password from standard input without prompting, or read the password from the console without echo.
- `-k` Deletes the key and password.

**Examples**

The following example adds the `admin_key` registry key:

```bash
wsetpkey -a admin_key
```
wsetpm

Enables or disables a profile manager to operate in dataless mode.

Syntax

wsetpm –d @ProfileManager:profile_manager

wsetpm –D @ProfileManager:profile_manager

Description

The wsetpm command specifies whether a profile manager runs in dataless mode. The dataless mode enables profile managers to distribute to endpoints that have no client database. Endpoints can subscribe to profile managers running in the dataless mode. A managed node can subscribe to any profile manager regardless of the distribution mode. If a dataless profile manager distributes to a managed node, the managed node database is ignored during the distribution.

Options

- d Enables the profile manager to operate in a dataless mode.

- D Disables the profile manager from operating in dataless mode. Before disabling dataless mode, you must remove all endpoint subscribers from the profile manager.

@ProfileManager:profile_manager

Specifies the name of the profile manager.

Authorization

admin, senior, super

Examples

1. The following example sets the AdminServer profile manager to operate in dataless mode. Endpoint subscribers are allowed.

wsetpm -d @ProfileManager:AdminServer

2. The following example sets the TopLevel profile manager to not accept endpoint subscribers. Dataless operation is disabled.

wsetpm -D @ProfileManager:TopLevel
wsetpr

Assigns the policy used in a policy region, enables or disables policy validation, and adds or removes a managed resource in a policy region.

Syntax

wsetpr [–d default_policy] [–v validation_policy] [-E | -e] resource region

wsetpr [-r] resource region

Description

The wsetpr command specifies which default or validation policy is used for the specified resource in the specified policy region. If the –E or –e option is used, this command enables or disables policy validation for the specified resource in the specified policy region. The wsetpr command also adds or removes a managed resource in a policy region. By default, the command adds the specified resource to the policy region. To remove a managed resource, use the –r option.

Options

–d default_policy
   Specifies the label of the default policy to be used for the managed resource.

–e
   Enables policy validation.

–E
   Disables policy validation.

–r
   Removes the specified resource from the policy region.

–v validation_policy
   Specifies the label of the validation policy to be used for the managed resource.

region
   Specifies the label of the target policy region.

resource
   Specifies the managed resource type.

Authorization

policy in addition to senior or super

Examples

1. The following example adds resource TaskLibrary to the Engineering policy region:
   wsetpr TaskLibrary @PolicyRegion:Engineering

2. The following example enables policy validation for the TaskLibrary resource in the Engineering policy region. The default policy is BasicTaskLibrary and the validation policy is BasicTaskLibrary.
   wsetpr -d BasicTaskLibrary -v BasicTaskLibrary -e 
   TaskLibrary @Engineering

See Also

wcrtpr, wdelpr, wgetpr
wsetquery

Edits the properties of a query.

Syntax

\texttt{wsetquery \[-n\] name \[-d\] description \[-r\] repository \[-v\] view \[-c\] column... \[-i\] \[-s\] \[-w\] where_clause \[-x\] query_name}

Description

The \texttt{wsetquery} command enables you to change information about an existing query. You can change the query name, description, repository, view, columns list, and the where clause.

Options

\texttt{\[-c\] column}

Changes the column or columns from which the query retrieves information. To include more than one column, use multiple \texttt{\[-c\]} clauses. The columns in the output are ordered according to how you enter them here.

\texttt{\[-d\] description}

Changes the description of the query.

\texttt{\[-i\]}

Reads a new where clause from standard input.

\texttt{\[-n\] name}

Changes the name of the query.

\texttt{\[-r\] repository}

Changes the repository from which the query gets information.

\texttt{\[-s\]}

Reads a new structured clause from standard input. The clause should be in the following format:

\texttt{[AND \ | \ OR] \ [NOT] \ column_name \ \{= \ | \ != \ | \ < \ | \ <= \ | \ > \ | \ >= \ | \ LIKE \ | \ IN\} \ column_value}

\texttt{\[-v\] view}

Changes the view or table that the query uses to retrieve information from the database.

\texttt{\[-w\] where_clause}

Reads a new where clause from the command line.

\texttt{\[-x\]}

Specifies that the output of the query does not contain duplicate rows.

\texttt{query_name}

Specifies the query to be changed.

Authorization

query\_edit, admin, senior, or super

Examples

1. The following example changes the where clause of the DOS-machines query to check the operating system version as well as the operating system name. It reads the new where clause from the command line.
wsetquery -w "BOOTED_OS_NAME = 'DOS' AND \n  BOOTED_OS_VERSION LIKE '6.%'" DOS-machines

2. The following example changes the name of the DOS-machines query to AIX-machines, changes the description of the query, and reads a new where clause from standard input.

wsetquery -i -n AIX-machines -d "Find all the AIX \nmachines" DOS-machines <<EOF
(BOOTED_OS_NAME = 'AIX')
EOF

See Also
wcrtdb|wcrquery|wgetquery|wrinvquery|wrquery
wsetrim

Edits the properties of an RDBMS interface module (RIM) object.

Syntax


Description

The wsetrim command changes the database information for a given RIM object. You can change the database ID, database user, database home, database server ID, and instance home. To change the vendor for a RIM object, you must delete the object using the wdel command and re-create it using the wctrim command. To change the managed node for a RIM object, you can either move the RIM object using the wmvrin command or delete and re-create the RIM object. To change the label for a RIM object, you can either use the wsetrim –n command or delete and re-create the RIM object.

Note: When you specify a path that contains a space, you must enclose the path name in quotation marks (" "). On Windows systems, you can also specify the MS-DOS path. For example, you can specify the path c:\Program Files\sqllib in one of the following ways:

- "c:\Program Files\sqllib"
- c:\progra~2\sqllib

Options

–a application_label
Changes the application label for the RIM object.

–d database
Changes the name of the database or data source to which the RIM object connects.

DB2      The DB2 database alias or the local DB2 database name.
Oracle    The name of the Oracle instance. This name is the SID option in the configuration connection file. Corresponds to the value in the ORACLE_SID variable.
Sybase    The name of the database that the application will use.
Microsoft SQL
          The name of the ODBC data source that RIM uses to connect to the Microsoft SQL Server database.
Informix  The name of the data source defined in the .odbc.ini file.

–H db_home
Changes the path to the database home directory. This option changes the value of the variables ORACLE_HOME, SYBASE, and DB2DIR for Oracle, Sybase, and DB2, respectively.

Note: On Windows operating systems, use one of the following formats when specifying value for a path that contains spaces:

- "c:\Program Files\sqllib"
• c:\progra~1\sql\lib

If you do not enclose the value within double quotation marks (""") or include the tilde (~) in the path name, the parsing of the command fails.

−I instance_home
For DB2 only, changes the path to the DB2 instance for the specified RIM object.

−m max_connections
Changes the maximum number of connections between the RIM object and the RDBMS.

−n name
Changes the name of the RIM object to name.

−s server_id
Changes the server ID for the database. This option changes the values of the variables TWO_TASK, DSQUERY, and DB2COMM for Oracle, Sybase, and DB2, respectively. For Microsoft SQL Server, this is the name of the RDBMS server machine.

−t instance_name
For DB2 only, changes the name of the DB2 instance for the specified RIM object.

−u user
Changes the name of the database user that the RIM object uses. If you are using DB2, specify a valid UNIX user.

rim_name
Specifies the label of the RIM object to be modified.

Authorization

senior or super in the Tivoli region

Examples

The following example changes the database ID to inventory, the database user to tivoli2, the database home directory to /ORACLE, and the database server ID to invdb2.world for the inventory RIM object.

wsetrim -d inventory -u tivoli2 -H /ORACLE \  
-s invdb2.world inventory

To verify the changes, use the wgetrim command:

wgettrim inventory

RIM Host: amon-sul
RDBMS User: tivoli2
RDBMS Vendor: Oracle
Database ID: inventory
Database Home: /ORACLE
Server ID: invdb2.world
Instance Home: 

Note that the output for the Instance Home field is blank; this field applies only to DB2.

See Also

wcrtrim, wgetrim, wsetrimpw
wsetrimpw

Sets the database password for an RDBMS Interface Module (RIM) object.

Syntax

wsetrimpw rim_name [old_password] [new_password]

Description

The wsetrimpw command sets the database password on a RIM object. The command prompts you for a password unless you specify the old_password and new_password options.

Options

new_password
  Specifies the new password.

old_password
  Specifies the current password.

rim_name
  Specifies the label of the RIM object.

Authorization

senior or super in the Tivoli region

Examples

The following example changes the password for the inventory RIM object from funEguy to Dlite:

wsetrimpw inventory funEguy Dlite

See Also

wcrtrim wgetrim wsetrim
wsettap

Sets the properties of the Tivoli Authentication Package on a Windows managed node.

Syntax

```
wsettap [-a | -d] [-B] [-P] [-r [domain_name\user_name | user_name]] [-k]
```

Description

The `wsettap` command sets the properties of Tivoli Authentication Package, TivoliAP.dll on the local managed node. The Tivoli Authentication Package enables Tivoli Management Framework to access remote file systems in the context of a user. The `wsettap` command also enables a Windows managed node to run setuid methods, that is, to run a method in the context of a user associated with the method. Refer to the *Tivoli Management Framework Planning for Deployment Guide* for more information about accessing remote file systems.

The Tivoli remote access account specifies a user account. Tivoli Management Framework uses this account to access remote file systems.

Using the `wsettap` command with no options prints version information from the currently running TivoliAP.dll.

When activating the Tivoli Authentication Package for the first time, which is usually immediately following Tivoli server installation, you must restart the machine for changes to take effect.

**Note:** In a Windows 2000 domain environment that uses Active Directory, Tivoli Authentication Package uses the Tivoli remote access account to connect to Active Directory rather than using an anonymous login. To use this feature, you must set the Tivoli remote access account to a valid domain-qualified user ID and password.

Options

- `–a` Sets the Tivoli Authentication Package internal key and registers with the local security authority (LSA). The new internal key becomes effective immediately. The TivoliAP.dll file is loaded by the LSA when the machine is restarted.

- `–B` Authenticates domain users using a domain controller other than the primary domain controller. To authenticate users using the primary domain controller, use the `–P` option.

- `–d` Removes the Tivoli Authentication Package internal key and unregisters the with the LSA. The file is released by the LSA (so that it can be deleted if Tivoli Management Framework is uninstalled) when the machine is restarted.

- `–k` Specifies that the password for `user_name` is read from standard input. If you do not specify this option, you are prompted for the password.

- `–P` Authenticates domain users using the primary domain controller. This is the default setting. To authenticate users using other domain controllers, such as backup domain controllers, use the `–B` option.
wsettap

-r [domain_name\user_name | user_name]
Sets the Tivoli remote access account to the specified user name. Tivoli Enterprise software accesses remote file systems using this user account. The specified user name can be prefixed with the domain name, separated by either a forward slash (/) or a backward slash (\). If the domain is specified, it must be the domain in which the machine running Tivoli Authentication Package belongs or a domain trusted by that domain. If no domain is specified, the operating system looks for the given user in the local domain or trusted domains. To see changes take effect immediately, restart the object dispatcher.

Authorization
Windows operating system permission to edit the registry.
To run the command without options: admin.

Examples
1. The following example disables Tivoli Management Framework access to remote file systems:
   wsettap -r ""
2. The following example sets the Tivoli remote access account to the userTME user account. The command reads the password from the passwd.txt file.
   wsettap -r userTME -k < passwd.txt
wsettask

Sets the properties of a task.

Syntax

```
  wsettask  -t task_name -l library_name [-g group_name] [-u user_name] -r role [-c comments] -i interp_type node_name file_name...
```

Description

The wsettask command sets the properties of a task in the specified task library.

Options

- **c comments**
  Adds any explanatory comments that help identify the task and its purpose.

- **g group_name**
  Specifies the name of the group under which the task runs. (UNIX only)

- **i**
  Defines the information required to execute the new task on a managed node. You must supply each of the following values:

    - **file_name**
      Specifies the name of the executable file to be run on the specified platform.

    - **interp_type**
      Specifies the interpreter type of the platform on which the task is to be run.

    - **node_name**
      Specifies the managed node containing the executable file for the specified platform.

- **l library_name**
  Specifies the task library in which the task resides.

- **r role**
  Specifies the authorization roles required to run the task. Multiple roles can be specified in a colon-separated list, for example admin:senior:super.

- **t task_name**
  Specifies the name of the task being updated.

- **u user_name**
  Specifies the name of the user under which the task runs.

Authorization

  admin, senior, super

See Also

  wcrtask, wdeltask, wgettask
wsetval

Adds or deletes a registry entry.

Syntax

wsetval [-b] [-d] [-h registry_hive] -k [key | @file_name] -n value_name -v ["value_string" | @file_name]

Description

On Windows endpoints only, the wsetval command adds or deletes a registry entry (key name) and sets its value (key value). If the key or value does not exist, it is created.

Note: This command should be run from an endpoint.

Options

- `-b` Creates the value as binary. Binary values must be read from a file specified with the `-v` option.
- `-d` Deletes a value name specified by the `-n` option or a key name specified by the `-k` option. If the `-n` option is present, the value name specified by the `-n` option is deleted. If the `-n` option is not present, the key (and all its subkeys) specified by the `-k` option is deleted.
- `-h registry_hive` Specifies the registry hive to update. Valid values are as follows:
  - HKEY_LOCAL_MACHINE (default)
  - HKEY_CURRENT_USER
  - HKEY_CLASSES_ROOT
  - HKEY_USERS
  - HKEY_CURRENT_CONFIG (Windows)
  - HKEY_DYN_DATA (Windows)
- `-k key | @file_name` Specifies the key in which the value is inserted. If the first character of the key is `@`, the key is read from `file_name`.
- `-n value_name` Specifies the name of the value. The `-n` option is optional if `-d` is present.
- `-v "value_string" | @file_name` Specifies the value or file that contains the value. The `-v` option is ignored if `-d` is present.

Authorization

admin

Examples

To add the NOTEPAD subkey under the existing SOFTWARE key, and assign the NOTEPADVAR key value name in the HKEY_LOCAL_MACHINE hive, enter the following command:

wsetval -h HKEY_LOCAL_MACHINE -k SOFTWARE\NOTEPAD
- -n NOTEPADVAR -v C:\TEMP\NTPADVAR.FIL
wsndnotif

Translates standard input into a message structure and sends it to the notification server.

Syntax

```
wsndnotif [-e] [-t] notice_group priority
```

Description

The `wsndnotif` command is a command line program that translates standard input into a message structure and sends it to the notification server. The `-e` and `-t` options enable shell scripts to send messages in a language-neutral format. If these options are not specified, this command treats standard input as a single ASCII buffer and the message is not translated into a language-neutral format. This command sends the message to the specified notice group using the specified priority.

Options

- `-e` Specifies that standard input is the ASCII representation of an exception, such as an exception generated from an Interface Definition Language (IDL) call.
- `-t` Specifies that standard input should be translated as a `tmf_msg_t` in the ASCII format used by the Extended Interface Definition Language (EIDL) shell method type. For more information about the ASCII format used by the EIDL shell method type, see `TME 10 ADE Framework Services Manual`.

`notice_group` Specifies which notice group to send the translated standard input.

`priority` Specifies the priority to use to send the translated standard input to the specified notice group. Valid priorities are `Critical`, `Error`, `Warning`, `Notice`, and `Debug`.

Authorization

`user`, `senior`, `super`

Examples

1. The following example sends a notice to the Tivoli Administration notice group. The priority is Notice.

   ```bash
   wsndnotif "Tivoli Administration" Notice
   This notice is to inform all administrators that I am changing the name of managed node homer to marge this weekend.
   
   Paul
   ```

2. The following example uses the `-e` option in a shell script. If an IDL call returns a nonzero exit status, the ASCII output is an exception. The ASCII data is logged to the Tivoli Administration notice group as an error.

   ```bash
   out=`idlcall $OID method`
   if [ ! -ne 0 ]
   then
   ```
3. The following example uses the -t option in a shell script. This example sends
message 49 from the task_msg message catalog to the Tivoli Administration
notice group. Strings Amar, root@indianola.tivoli.com, and AmarLib are
inserted into the message. If message 49 cannot be found, the default message
is sent. The default message is "A new task, %1$s, was created by %2$s in
the %3$s task library".

```sh
wsndnotif -e "Tivoli Administration" Error <<EOF
$out
EOF
fi
```

See Also

- wexpnotif
- wlsnotif
- wtailnotif
**wstandalone**

Creates a managed node without requiring the Tivoli server to initiate a connection to the local machine.

**Syntax**

```
wstandalone -c tivoli_server [-i] [managed_node]
wstandalone -a managed_node dispatcher policy_region
```

**Description**

The `wstandalone` command enables you to create a managed node without requiring the Tivoli server to initiate a connection to the local machine. This command is useful for situations in which establishing a connection would create a security risk.

*Note:* The `wclient` command enables you to create a managed node using an encrypted SSH connection. For more information, see Chapter 103.

To create a managed node using `wstandalone` command, follow these steps:

1. Create a Tivoli server on the machine on which you want to create a managed node. Set the SetPort parameter for this Tivoli server to 1 so that remote start capability is enabled. See the *Tivoli Enterprise Installation Guide* for more information about creating a Tivoli server and setting the SetPort parameter.

2. Convert the Tivoli server you created in step 1 to a managed node using the following command, where `tivoli_server` is the name of the Tivoli server that you want the managed node to connect to, and `managed_node` is an optional string that you specify only if you want to change the default label for the managed node:

   ```bash
   wstandalone -c tivoli_server managed_node
   ```

3. Run the following command on the Tivoli server, where `managed_node` is the name of the managed node you created in step 2. `dispatcher` is the object dispatcher number of the managed node, and `policy_region` is the policy region to which you want to add the managed node.

   ```bash
   wstandalone -a managed_node dispatcher policy_region
   ```

**Options**

- **-a** Adds the managed node to the specified policy region and registers the managed node with the Tivoli server. Before you run this command, you must create the managed node and specify a Tivoli server using the `wstandalone` -c command. You must run the `wstandalone` -c command on the Tivoli server.

- **-c** Creates a managed node on the local machine and adds the managed node to the list of managed nodes for the specified Tivoli server. Before you run this option, you must install a Tivoli server on the local machine. This option converts the Tivoli server on the local machine to a managed node. The `managed_node` parameter is optional with the `wstandalone` -c command. Specify the `managed_node` value only if you want to change the default label for the managed node. The default label for the managed node is the output of the `hostname` command. You must run the `wstandalone` -c command on the machine on which you want to create a managed node.
Causes the command to prompt for the installation password. If you do not use this option, there is not a password prompt.

dispatcher
Specifies the object dispatcher number of the managed node as identified using the odadmin odlist command.

managed_node
Specifies the label of the managed node.

policy_region
Specifies the policy region to which the managed node is to be added.

tivoli_server
Specifies the host name of the Tivoli server to which this managed node is to be connected.

Authorization
To run the wstandalone –c command, you must have root access on the local machine. On Windows systems, you must be a member of the Windows Administrators group.

To run the wstandalone –a command, super or install_client is required.

Examples
1. The following example creates a managed node on the local machine and specifies that the managed node is to connect to the Tivoli server oak.
   wstandalone -c oak
2. The following example creates a managed node on the local machine, specifies that the managed node is to connect to the Tivoli server oak, and changes the label of the managed node to birch.
   wstandalone -c oak birch
3. The following example adds the managed node birch to the policy region my_region.
   wstandalone -a birch 50 my_region

See Also
odadmin, wclient, winstlcf, wserver
wstartsched

Starts the Tivoli scheduler.

Syntax

wstartsched

Description

The wstartsched command starts the Tivoli scheduler, the TMF_sched service.

Authorization

senior

See Also

wdelsched  wedsched  wenblsched  wgetsched  wschedjob
wsub

Subscribes Tivoli resources to a profile manager.

Syntax

```
wsub [-r] name subscriber...
```

Description

The `wsub` command subscribes the Tivoli resources specified in `subscriber` to the profile manager specified in `name`.

Options

- `-r` Specifies that the `wsub` command returns an error code (1) if any of the subscribers are unreachable. A failure code is always returned if the specified subscriber does not exist.

- `name` Specifies the name of the profile manager to which to subscribe the resources. Valid formats for the `name` option are as follows:
  - `@prof_manager_name`
  - `@ProfileManager:prof_manager_name`
  - `/Regions/PolicyRegionName/prof_manager_name`

- `subscriber` Specifies the names of the Tivoli resources to add to the subscription list of the specified profile manager. This option can be specified multiple times. The following are examples of valid formats for the `subscriber` option. If you are subscribing a resource other than a managed node, modify these examples to reflect your resource type.
  - `@ManagedNode:node_name`
  - `/Regions/PolicyRegionName/node_name`

Authorization

`admin` in the policy region of the profile manager and the policy region of the subscriber.

Examples

The following example subscribes the managed nodes pinatubo and newcastle and the profile manager Apps_Dev to the Development profile manager:

```
wsub @Development @ManagedNode:pinatubo \
   @ManagedNode:newcastle @ProfileManager:Apps_Dev
```

See Also

`wcrtprf`, `wcrtprfmgr`, `wdistrib`, `wgetprf`, `wgetsub`, `wlssub`, `wpopulate`, `wunsub`, `wvalidate`
wsupport

Collects problem information from users to send to a customer support representative. (UNIX only)

**Syntax**

```
wsupport -s
```

**Description**

The `wsupport` command prompts you for essential information needed by a support representative to investigate technical problems. After the information is entered, you are prompted whether the information is e-mailed to the support representative or is saved to a text file. If you specify e-mail, you must enter the e-mail address to which you want to send the information. The provided information is mailed in a format that can be parsed, one item per line. If you choose to save the information to a text file, the information is presented in an ordered format that can be faxed to a support representative. You can edit the information before faxing it. You can also e-mail the file at a later time. If you are going to e-mail the file to a support representative later, it is recommended that you choose the e-mail option when prompted, and then specify No on the final confirmation.

You are prompted for the following information:

**Name**

Name of the person submitting the request.

**Company Name**

Name of the company.

**E-mail Address**

E-mail address of the submitter, if e-mail is available.

**Telephone Number**

The telephone number of the submitter. A telephone number is required when an e-mail address is not provided.

**Fax Number**

The fax number of the submitter.

**TMR Number**

The number of the licensed Tivoli region.

**Support E-mail Address**

The e-mail address of the support provider used by the company.

You are prompted for the following system information:

**Call ID number**

The call tracking number of an existing problem logged by your support provider.

**System Type**

The system on which the problem occurred.

**Operating System Release**

The release level of the operating system on which the problem occurred, such as 4.1.3.
wsupport

Tivoli Product
The name of the Tivoli product in which the problem occurred. 
wsupport displays a list of Tivoli products from which to choose.

Tivoli Release
The version and release number of the product.

You are prompted for the following problem information:

Where the problem occurred
Whether the problem occurred on the Tivoli server or a managed node.

Name of server or client
Name of the system on which the problem occurred.

Severity of the problem
Severity level of the problem. Valid options are as follows:
- **Critical**—Customer cannot conduct business, product is inoperative, loss of operations, continuous failures or interruptions, data corruption.
- **Major**—Systems or application interrupted but recovered, high risk of reoccurrence, urgent, intermittent failures, significant performance degradation.
- **Minor**—Problem encountered, irritant, minimal impact to business operations, localized or isolated impact, operational nuisance.
- **No Impact**—General questions or information needed.

Short description
Ten to twelve word description of the problem.

Attempted solutions
Actions taken that did not solve the problem.

Long description
Detailed description of the problem.

The problem information can be saved to the .tivoli_rc file in the user home directory.

Options

- `-s` Skips the customer information questions. Prompting begins with the system information questions. Customer information is read from the .tivoli_rc file.

Files

- `/tmp/wsupport.$UNAME`—Generated report file.
- `/tmp/sup.$UNAME.uu`—Encoded compressed log files.
wtailnotif

Displays notices as they are posted to notice groups.

Syntax

wtailnotif [-a administrator...] [-g notice_group...] [-l] [-p priority...]

Description

The wtailnotif command connects to the notification server and displays new notices as they are posted. You can filter the notices by priority, administrator, or group. You can also specify whether you want to display all notice data or only the notice headers. When you do not specify options, all notices from all notice groups are displayed as they are received.

Options

-a administrator
   Specifies that only notices posted by the specified administrator are to be displayed. Multiple administrators can be specified.

-g notice_group
   Specifies that only notices posted to the specified notice group are to be displayed. Multiple notice groups can be specified.

-l
   Specifies that only notice headers are to be displayed.

-p priority
   Specifies that only notices with the specified priority are to be displayed. Multiple priorities can be specified.

Authorization

senior, super

Examples

The following example displays notices as they are posted to the TME Administration notice group:

wtailnotif -g "TME Administration"

See Also

wexpnotif, wlsnotif, wsndnotif
**wtaskabort**

Stops a task transaction and rolls back any uncommitted changes.

**Syntax**

```sh
wtaskabort
```

**Description**

The `wtaskabort` command is for use within a task shell script. You cannot use this command from the command line to cancel a specific task.

When tasks are processed, a transaction model can be specified. Using Tivoli commands, it is possible to perform multiple operations as part of one transaction. Any change does not become permanent until the entire task completes and the transaction is committed. If a failure occurs in the task with a non-Tivoli command, the `wtaskabort` command can be used to stop and cancel the transaction and roll back all changes that have not been committed.

**Notes**

When the `wtaskabort` command is used, the task does not return any output or return code. Instead, an error message similar to the following is displayed:

```sh
bald (ManagedNode): The task failed to execute.
bald (ManagedNode): System Exception: failure detected by object adapter: completion status: NO
Transaction Error
```

**Examples**

In the following example, the task registers a special resource in the Tivoli name registry, which stores the path to a directory. Then, the directory is created. If the `mkdir` command fails, the script issues the `abort()` function because the shell is running with the `-e` option. In the `abort()` function, `wtaskabort` is called, which rolls back the change made to the name registry. The special resource does not show up in any subsequent lookups.

```sh
#!/bin/sh
set -e

# Function to be used to abort a transaction in task shell script
#abort()
{
    return_code=$?
    if [ $return_code -ne 0 ]; then
        wtaskabort
        fi

trap 'abort' 0

# TASK MAIN
#wregister -i -r special_directory /Tivoli/specials OBJECT_NIL
mkdir /Tivoli/specials
```

See Also

wruntask | wrunjob | Tivoli transactions
wtimezone

Prints the time zone value for the specified system.

Syntax

wtimezone node_name

Description

The wtimezone command prints the time zone by printing the number of minutes west of Greenwich mean time (GMT).

Options

node_name

Specifies the name of the host.

Authorization

user, admin, senior, super

Examples

The following example shows the time zone of the managed node bald:

wtimezone bald

360

See Also

wdate, wdiskspace, whostid, wifconfig, winstdir, winterp, wmannode, wmemszie, wping, wuname, wxterm
wtemp

Displays the name of the directory where Tivoli Enterprise products create temporary files.

**Syntax**

```
wtemp [-s]
```

**Description**

The `wtemp` command displays the name of the directory where Tivoli Enterprise products create temporary files and confirms that this directory exists and has write permissions.

`wtemp` uses the forward slash (/) in all path names. On Windows operating systems when you are not using the bash shell, you need to convert backslashes (\) to forward slashes (\).

**Options**

- `-s` Confirms that the directory exists and has write permissions.

**Authorization**

No Tivoli authorization role is required.

**Return Values**

If the temporary directory does not exist or cannot be written to, it writes a null string to standard output.

**Examples**

1. The following example displays the temporary directory on a Solaris operating environment:
   
   ```
   wtemp /var/tmp
   ```

2. The following example displays the temporary directory on a Windows operating system:
   
   ```
   wtemp c:/Tivoli/db/cdeamqs.db/tmp
   ```

3. The following example confirms that the temporary directory on an AIX system exists and has write permissions:
   
   ```
   wtemp -s /tmp
   ```

4. The following example shows that the temporary directory either does not exist or does not have write permissions:
   
   ```
   wtemp -s
   ```
Imports and exports task library definitions.

**Syntax**

```
wtll [-r] -p region [-P preprocessor] import_file [preprocessor_options...]
wtll -i [-p region | -l library_name] [-t task...] [-P preprocessor] import_file [preprocessor_options...]
wtll -d [-l library_name] [-P preprocessor] import_file [preprocessor_options...]
wtll -F export_file -l library_name
```

**Description**

The `wtll` command is a tool used to import (install) task library definitions and create or modify Tivoli task libraries. The command is also used to export existing task libraries in a form that allows them to be saved offline or transferred to other installations. Task libraries are described with the Task Library Language (TLL) for both import and export purposes.

When importing a new task library into a Tivoli region, a TLL source file is prepared. All programs or scripts to be used by the tasks are either included directly in the TLL source or are referenced as external files by TLL directives. The `wtll` command reads and parses the task library definition, validates the various attributes defined in the source, and then either creates or modifies a task library object.

When exporting a task library, `wtll` creates a tar-format file that contains all materials needed to reconstruct the task library. The collected export files include a TLL description of the tasks along with the code of each task for each platform type supported.

For more information about using the `wtll` command to create task libraries, see Part 3, “Task library language,” on page 433.

**Options**

- **-d** Runs the command in debug mode, which checks the syntax of imported file, but does not import the file into the Tivoli region.
- **-F export_file** Creates a tar-format export file of the task library specified by `library_name`.
- **-i** Inserts a new task or group of tasks into an existing task library. The import file is parsed, and specified tasks are created in the existing task library. The import file can be a complete task library or a list of individual tasks.

**Notes:**

- Use the `wcrttask` command to add a task to a task library.
- If the import file contains a task library, the task contained must be self contained. The task cannot use the layout keyword to specify an ArgLayout defined in the task library.
-l library_name
   Specifies the name of the task library for either exporting or for modifying. The library must be defined in the local Tivoli region.

-p region
   Specifies the policy region in which to create the new task library. The policy region must exist within the local Tivoli region.

-P preprocessor
   Specifies the path to the program to use as a preprocessor on the import file before it is parsed. The cpp command is the mostly commonly used preprocessor. Any options listed on the command line after import_file are assumed to be options to be passed to the preprocessor (preprocessor_options).

-r
   Replaces a task library with the library specified in import_file. The existing library and all its tasks and jobs are deleted, and then the library is re-created with the features specified in the import file. If the library does not already exist, a new one is created.

t task
   Specifies the name of the task to import. The specified task name must be unique within the task library or policy region.

import_file
   Specifies the file to import. This file can contain a single task to be added to a task library or an entire new task library, which replaces the existing task library.

preprocessor_options
   Specifies additional preprocessor options. Refer to the documentation for the preprocessor for valid options.

Authorization
   admin, senior

Examples
   1. The following example creates a new task library in the policy region sandia-Region using the TLL source in the /tmp/tll file:
      wtl1 -p sandia-Region /tmp/tll

   2. The following example also creates a new task library in the policy region, but it first runs the TLL source through the cpp preprocessor and uses include files from the user’s home directory. The -B and -P options are input to cpp preprocessor. They parse C++ comments in the task library file.
      wtl1 -p sandia-Region -P /usr/lib/cpp /tmp/tll -B -P

      Note: The options for the preprocessor can use the same option names as those used for wtl1 (such as -P). Because of their location in the syntax statement, the command processes the options correctly.

   3. This example replaces the task library my_tasks, which is in the sandia-Region policy region, with the TLL source in the /tmp/tll file:
      wtl1 -r -p sandia-Region -l my_tasks /tmp/tll

   4. This example exports the task library my_tasks to the /tmp/my_tasks.tar file:
      wtl1 -F /tmp/my_tasks.tar -l my_tasks

See Also
   cpp, tar, wcrttask
wtmurname

Displays or changes the name of the local Tivoli region.

Syntax

```
wtmurname [-s new_name]
```

Description

The `wtmurname` command displays the name of the local Tivoli region. To change the name of the local Tivoli region use the `–s` option. In this case, all Tivoli regions must be connected, so that the new name can be verified as unique throughout the Tivoli environment.

Options

```
–s new_name
```

Changes the name of the local Tivoli region to the specified name. Before changing the name, the new name is verified against all connected Tivoli regions.

Authorization

The caller must have these roles in the local Tivoli region:

- To change the name: `super`
- To display the name: `super`, `senior`, `admin`, or `user`

Examples

1. The following example displays the name of the local Tivoli region:
   ```
   wtmurname
   sherman-region
   ```

2. The following example changes the name of the local Tivoli region to `patton-region`:
   ```
   wtmurname -s patton-region
   ```

See Also

```
wconnect  wdisconn  wlsconn  wupdate
```
wtrace

Provides information to help debug method calls.

Syntax

–k db_dir

Description

The wtrace command is used to diagnose problems in custom methods and executable files by examining method input, transactions, and method output. To use this command, you must enable tracing in the Tivoli object dispatcher. Enable tracing causes the creation of the odtrace.log log file. This file is created in the database directory of the specified managed node. By default, the trace log is 1 MB (512 2-KB-trace records), but this limitation can be changed by restarting the object dispatcher and specifying a new size with the –t option.

Tracing is persistent across invocations of the object dispatcher. This means that if the object dispatcher restarts, you do not have to re-enable tracing. Because this command examines the trace log directly, it does not require the object dispatcher to be active, although having an object dispatcher available to run an odstat command can be helpful.

The wtrace command has numerous formatting options that provide varying amounts of displayed data, but there following general rules apply to the output format:

• Structure output is enclosed in curly braces ({}).
• Arrays are enclosed in square brackets ([]).
• Sequences are structures that have a count field followed by an array of elements.
• A top-level transaction has the following general format:
  
  { }
  
  #4

The most common usage is as follows:

wtrace -jk /usr/Tivoli/spam.db

Output similar to the following is displayed:

loc-ec 676 15:10:36 M-H 1-289 0 NOT_FOUND
Object ID: 333333.1.387#FpPol::FilePackagePolDef#
Method: o_setattr
Method Args: fp_def_src_host
Principal: root@albundy (0/0)
Path: o_setattr
Trans Id:
  
  { }
  
  #3
In this output, you can determine that there was a local object call on thread ID 676 run at 15:10:36 local time. The `o_setattr` method was invoked on object reference 333333.1.387#FpPol::FilePackagePolDef# with the option `fp_def_src_host`. Error status of NOT_FOUND was returned. In this verbose form, the transaction ID, the principal invoking the method, the effective user ID and group ID of the method, and the path to the method are also provided.

The following output shows an error condition (exception) that was detected:

```
loc-ec 6073 16:28:01 M-hdoq 1-6047 26 e=12
Object ID: 333333.1.26
  Method: get_all
  Principal: root@ajax (60001/60001)
  Helper pid: 2419
  Path: /home/Tivoli/bin/solaris2/TMF/BASESVCS/TNR_prog1
  Trans Id:
    [333333:1,333333:1,7:4042]
    [333333:1,333333:1,7:4063]
  #3

Input Data: (encoded): "NisDomain" 9999
Results: (encoded):
  "Exception:UserException:SysAdminException::ExException:
    SysAdminException::ExInvalid:SysAdminException::ExNotFound"
  "The resource type %7$s was not found."
    779578081
    0
  "NisDomain"
```

In this output, you can determine that administrator root@ajax invoked the `get_all` method on the name registry (object reference 333333.1.26) with the effective user ID nobody (60001). Input Data shows that the `NisDomain` option was passed to the method. From the error code (e=12) and the output in Results, you can see an exception was detected indicating that a specific resource of type NisDomain was not found.

**Options**

- `-D` Prints large blocks of input data.
- `-E` Suppresses printing of error records.
- `-h` Prints header used in the “old” format.
- `-H` Suppresses printing of hex dumps.
- `-I` Suppresses printing of input records.
- `-j` Prints the preferred format for normal screens (80 columns).
- `-J` Prints the preferred format for wide screens (132 columns).
- `-k db_dir` Specifies the Tivoli database directory.
–l  Prints the long form output.
–n  Prints additional new lines between transactions.
–O  Suppresses printing of output records.
–u  Prints the usage statement.
–v  Prints in verbose mode.
–V  Prints the version number of the command.

**format_options**

Specifies additional options that modify formatting of the output. The available formatting options are as follows:

–e lines  Specifies the threshold for line breaks. The default is 5.
–f  Turns off printing of abbreviated numbers (for example, 3.2K instead of 3219).
–t tab_size  Specifies the tab increment. The default is four characters.
–w width  Specifies maximum width of the formatted line. The default is 80.
–W width  Specifies minimum width of the formatted line. The default is 70.

**Examples**

1. The following example enables tracing for all object calls on the current object dispatcher:
   odadmin trace objcalls
2. The following example enables tracing for all object calls on all object dispatchers:
   odadmin trace objcalls all
3. The following example enables tracing for error conditions on the current object dispatcher:
   odadmin trace errors
4. The following example enables tracing for object service calls on all object dispatchers:
   odadmin trace services all
5. The following example enables tracing of object service calls and errors on all object dispatchers:
   odadmin trace services all
   odadmin trace errors all

**Defects**

A corrupt trace log can cause the wtrace command to dump core.

**See Also**

odadmin
wuname

Lists operating system information.

Syntax

\texttt{wuname node\_name}

Description

The \texttt{wuname} command lists the operating system for the specified managed node. The command returns the operating system, version number, release number, and the hardware name.

Options

\textit{\texttt{node\_name}}

Specifies the managed node.

Authorization

user, admin, senior, super

Examples

The following example shows the operating system information for the managed node \texttt{bald}:
\begin{verbatim}
  wuname \texttt{bald}
\end{verbatim}

\texttt{SunOS bald 5.3 Generic_101318-21 sun4m}

See Also

\texttt{wdate, wdiskspace, whostid, wifconfig, winstdir, winterop, wmannode, wmemsize, wping, wxterm}
wuninst

Uninstalls Tivoli Enterprise products from a specified node or from the entire Tivoli region.

Syntax

wuninst

wuninst tag

wuninst –list

wuninst tag –list

wuninst tag node_name [–rmfiles] [options]

Description

The wuninst command is a wrapper script that invokes product-specific uninstall scripts. This section provides general usage information about this command. The usage statement for this command differs for each product.

To view the general usage statement, enter the following command:

wuninst

To view the usage statement for a specific Tivoli Enterprise product, enter the following command:

wuninst tag

where tag is the registered product tag.

To view a list of the product tags, enter the following command:

wuninst -list

If you include the node_name option, the Tivoli Enterprise product specified with the tag option is removed from the specified node. If node_name is the Tivoli server, the product is removed from the entire Tivoli region. You are prompted with a confirmation message before the product is removed from the Tivoli region.

You should use the wuninst command to remove all application files before running the wunstmn command to uninstall the managed node.

Options

–list  Lists the installed application tags or the managed node where a product is installed. If used with the tag option, the –list option lists the managed nodes where a specific product is installed. Without other options, the –list option shows the tags of all installed products.

–rmfiles  Indicates that all product files are to be removed from specified managed nodes. When you do not specify this option, the command removes the database entries only. When you issue this command from the Tivoli server and specify this option, all entries for each managed node in the Tivoli region is removed.
wuninst

node_name
   Specifies the name of the managed node from which to remove the product. If you specify the Tivoli server, the product is removed from all managed nodes in the Tivoli region.

options
   Indicates options that might be required by each product. To view the options required to uninstall a particular product, use the `wuninst tag` command.

tag
   Specifies the registered product tag for the Tivoli Enterprise product to remove.

Authorization
   super

Examples
   1. The following example returns the general usage statement of the command:
      
      `wuninst`
   2. The following example lists the registered product tags of all Tivoli Enterprise products installed in the Tivoli region:
      
      `wuninst -list`
   3. The following example lists the managed nodes where Tivoli Software Installation Service, Version 3.6, is installed:
      
      `wuninst SIS_3.6 -list`
   4. The following example removes Tivoli Software Installation Service from managed node kiwi.
      
      `wuninst SIS_3.6 kiwi`
   5. The following example removes Tivoli Software Installation Service from managed node pctmp83. Because this managed node is the Tivoli server, Tivoli Software Installation Service is removed from every node in the Tivoli region. The `-rmfiles` option indicates that all product files are removed as well as database entries.
      
      `wuninst SIS_3.6 pctmp83 -rmfiles`

See Also
   `wunstmn`
wunstmn

Removes Tivoli Management Framework files from a UNIX or Windows managed node.

Syntax

wunstmn [-A] [-f] [-r] [-y] [...] [name [...]]

Description

The wunstmn command uninstalls the Tivoli Management Framework files from specified UNIX and Windows managed nodes. Removing these files removes the managed node from the Tivoli region.

To run this command, the managed node must be connected to the Tivoli server and the command must be run from the $BINDIR/TAS/UNINST directory. If the managed node has lost its connection to the Tivoli server, specify the –r option.

Note: You cannot uninstall the Tivoli server or the managed node on system where you issue this command.

To remove a managed node that has Tivoli Enterprise applications installed, use the wuninst command to remove the applications before using the wunstmn command to remove the managed node.

When you do not specify the –A option, the wunstmn command removes only the client database. When you specify the –A options, the wunstmn command removes all files associated with Tivoli Management Framework.

Note: Do not use the –A option if the managed node to be removed shares files with other managed nodes that you want to keep in your Tivoli environment.

After this command removes the installed files, it calls the wrmnode command to remove all entries about this node from the remaining Tivoli databases. After the wunstmn command completes, you should run the wchkdb command to synchronize the object databases.

Options

-A Removes all Tivoli Management Framework files from the specified managed node. This includes libraries, binaries, manual pages, message catalogs, and so on. Without this option, the command removes only the client database from the specified managed node.

-f Specifies that the name option is a file name containing a list of managed nodes to be uninstalled. The file format is as follows:

ManagedNode_name user_name

-r Uses rexec as a communication protocol. Use this option to uninstall a managed node when the object dispatcher is not running.

-y Suppresses confirmation. This option allows wunstmn to run unattended on multiple managed nodes.

name Specifies either the name of a single managed node to uninstall or the name of a text file containing multiple managed nodes to uninstall. If used
wunstmn

with the –f option, name specifies the path to a text file. You can specify multiple file names. Without the –f option, name is the name of a single managed node.

Authorization
super

Examples

1. The following example removes Tivoli Management Framework and all files from managed node iandu-4:
   wunstmn -A iandu-4

2. The following example uses a file named nodelist to uninstall a group of managed nodes. The –y option suppresses the confirmation. Because –A is omitted, only the databases are removed. All Tivoli Management Framework files remain on the nodes.
   wunstmn -f -y nodelist

See Also
wchkdb wrmnode wuninst
wunsub

Removes Tivoli resources from the subscription list of a profile manager.

Syntax

\texttt{wunsub} [\texttt{-a}] [\texttt{-l}] [\texttt{-r}] \texttt{name} [\texttt{subscriber...}]

Description

The \texttt{wunsub} command removes the specified Tivoli resources from the subscription list of the specified profile manager.

When the \texttt{-l} option is specified, any configuration information distributed to subscriber databases from this profile is maintained in the subscriber database, but the information is made local to the subscriber. When the \texttt{-l} option is not specified, configuration information in the subscriber database that was distributed from this profile is deleted from the subscriber database.

When the \texttt{-a} option is specified, all current subscribers are removed.

Options

\texttt{-a} \hspace{1em} Removes all current subscribers to the specified profile manager.

\texttt{-l} \hspace{1em} Specifies to maintain the subscriber database, but make it local.

\texttt{-r} \hspace{1em} Specifies that the command returns an error code (1) if any of the subscribers are unreachable. A failure code is always returned if the specified subscriber does not exist.

\texttt{name} \hspace{1em} The name of the profile manager from which to unsubscribe the resources. Valid formats for are as follows:

\begin{itemize}
  \item \texttt{profile_manager_name}
  \item \texttt{@ProfileManager:profile_manager_name}
  \item \texttt{/Regions/policy_region_name/profile_manager_name}
\end{itemize}

\texttt{subscriber...} \hspace{1em} The names of the Tivoli resources to remove from the subscription list of the specified profile manager. This option can be specified multiple times. Valid formats are as follows:

\begin{itemize}
  \item \texttt{@node_name}
  \item \texttt{@ManagedNode:node_name}
  \item \texttt{@Endpoint:endpoint_label}
  \item \texttt{/Regions/policy_region_name/node_name}
\end{itemize}

Authorization

\texttt{super, senior, admin}

Examples

1. The following example unsubscribes all subscribers of profile manager pm1, maintaining configuration information in the subscriber databases as local:

\texttt{wunsub -a -l pm1}
2. The following example unsubscribes subscribers pm2 and mn1 from profile manager pm1. All configuration information is deleted from the subscriber databases.

wunsub pm1 @ProfileManager:pm2 @ManagedNode:mn1

See Also

wcrtpf wcrtpfmg r wdistrib wgetprf wgetsub wlsub wpopulate wsub wvalidate
wupdate

Updates resources in the local name registry.

Syntax

wupdate [-f] -r resource [-r resource...] regions...

Description

The wupdate command updates resources in the local name registry from one or more remote Tivoli regions. When this command is run, resources are locked in the name registry. In some cases, the resource might already be locked, such as when another wupdate command is running. This command attempts to lock a resource for 60 seconds before timing out.

Options

- `f` Forces the update regardless of time stamp.
- `r resource`... Specifies one or more resource types to be updated. You can specify the resource type or use All to update all resource types.
- `regions`... Specifies one or more Tivoli regions from which to update. You can specify the region name or use All to update all regions.

Authorization

senior or super in the Tivoli region

Examples

1. The following example updates the local name registry with the ManagedNode resource type from the ceridwen-Region:
   wupdate -r ManagedNode ceridwen-Region
2. The following example updates the local name registry with all resource types from the ceridwen-Region and the meiron-Region:
   wupdate -r All ceridwen-Region meiron-Region
3. The following example updates the local name registry with the ProfileManager and AdministratorCollection resource types from all connected regions:
   wupdate -r ProfileManager -r AdministratorCollection All
4. The following example forces the update for all resource types regardless of the time stamps on the resource types:
   wupdate -f -r All meiron-Region

See Also

wconnect, wdisconn, wlsconn
wvalidate

Validates a profile against its validation policy.

Syntax

```
wvalidate name
```

Description

The `wvalidate` command validates the profile identified by `name` against its validation policy.

Options

```
name   The profile whose policy is to be validated. Valid formats are as follows:
  • @prof_name
  • @ProfileManager:prof_name
  • /Regions/PolicyRegionName/prof_manager_name/prof_name
```

Authorization

```
super, senior, admin
```

Examples

The following example validates the profile pr1 against the validation policy:
```
wvalidate @TestProfile:pr1
```

See Also

```
wcrtpfrf wcrtprfmgr wdistrib wgetprf wgetsub wlssub wpopulate wsub
wunsub
```
wxterm

Opens an Xterminal window on a specified display.

Syntax

wxterm –h node_name [xterm_options]

Description

The wxterm command opens an Xterminal window on a specified managed node. You can also supply configuration options as you would if you invoked the xterm program directly. If you do not supply configuration options, this command uses the default configuration options located in the install_dir/bin/interp/TAS/xterm.sh script.

Options

- h node_name
  Specifies the managed node where the xterm program is run.

  xterm_options
  Specifies options that are passed to the xterm program (such as default font, colors, and so on).

Authorization

  senior, super

Chapter 3. Commands alphabetically  393
## Part 2. Policy methods

### Chapter 4. Endpoint policy scripts  \(.\) . \(.\) \(.\) 397
- Endpoint policy methods \(.\) . \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) 397
- allow_install_policy \(.\) . \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) 398
- after_install_policy \(.\) . \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \(.\) \ }.
Chapter 4. Endpoint policy scripts

Tivoli Management Framework is installed with empty endpoint policy scripts. Endpoint policy scripts can be any type of executable file that accepts command line options, exits with a return code, and sends output to standard output. In most cases, a shell script is sufficient. Although some situations can require a more flexible programming language, such as Perl or C. The run time of these scripts (excluding the login_policy script) affects the amount and efficiency of logins that the endpoint manager can process at one time.

The following table contains the endpoint policy scripts and a description of where and when the script is run.

<table>
<thead>
<tr>
<th>Policy Name</th>
<th>Origin</th>
<th>When Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_install_policy</td>
<td>Run by the endpoint manager.</td>
<td>When the installation of the endpoint begins.</td>
</tr>
<tr>
<td>after_install_policy</td>
<td>Run by the endpoint manager.</td>
<td>After the existence of the endpoint is recorded by the endpoint manager and before the endpoint receives its initial login information.</td>
</tr>
<tr>
<td>login_policy</td>
<td>Run by the gateway.</td>
<td>Each time the endpoint logs in.</td>
</tr>
<tr>
<td>select_gateway_policy</td>
<td>Run by the endpoint manager.</td>
<td>Each time an endpoint needs to be assigned to a gateway.</td>
</tr>
</tbody>
</table>

Endpoint policy methods

This section describes the endpoint policy methods.
allow_install_policy

Validates whether an endpoint should exist in the current Tivoli region.

Syntax

allow_install_policy endpoint_label endpoint_object_id endpoint_interpreter_type
  gateway_object_id endpoint_ip_address region dispatcher version unique_id protocol

Description

The allow_install_policy script validates whether an endpoint should exist in the current Tivoli region. The default behavior of this policy allows endpoints to log in as long as the endpoint label is valid and not in use (the default is exit 0). You can also use this script to perform other processes before the endpoint is logged in. Remember, though, that the endpoint does not exist at the time this script is run. Therefore, the script cannot run Tivoli commands.

If the script fails, the installation process is terminated and no information is written to the Tivoli database.

The endpoint manager runs allow_install_policy when it receives the initial login packet for an endpoint from a gateway. The script is not run on subsequent logins. allow_login_policy is run before the select_gateway_policy script.

Options

dispatcher
  Specifies the object dispatcher number of the endpoint.

endpoint_interpreter_type
  Specifies the interpreter type of the endpoint.

endpoint_address
  Specifies the Internet Protocol (IP) address and port number of the endpoint.

  Note: When hostname resolution is enabled using the wepmgr
  ep_hostname_resolution command, this value can be either a
  hostname or an IP address.

deployer_label
  Specifies the label of the endpoint trying to log in.

deployer_object_id
  Specifies the object ID of the endpoint.

gateway_object_id
  Specifies the object ID of the intercepting gateway.

protocol
  Specifies the network protocol used by the endpoint logging in.

region
  Specifies the region number in which the endpoint resides.

unique_id
  Specifies the unique ID (also called the inventory ID) of the endpoint
  logging in.

version
  Specifies the current version of the endpoint software.
Environment Variables

These environment variables are automatically set when this script is run by the endpoint manager:

**LCF_DUPL_GATEWAY**
Specifies the object ID of the existing endpoint gateway.

**LCF_DUPL_INTERP**
Specifies the interp of the existing endpoint.

**LCF_DUPL_INV_ID**
Specifies the inventory ID of the existing endpoint.

**LCF_DUPL_LOGIN**
Specifies the timestamp of the existing endpoint first normal login.

**LCF_DUPL_NET_ADDRESS**
Specifies the network address of the existing endpoint.

**LCF_DUPL_OBJECT**
Specifies the object ID of the existing endpoint.

**LCF_INVALID_LABEL**
Specifies whether the endpoint label is valid according to test_label option of the wepmgr command.

Examples

The following is an example of the allow_install_policy script. This example does not allow endpoints on the subnet 146.84.26 to log in to the Tivoli management region. It also does not allow endpoints that have the string dev in their name.

This script is intended for UNIX operating systems. For Windows operating systems, the awk utility does not support all the capabilities of the awk utility on UNIX. In particular:

```bash
awk '{S=":" ; print $2}''
```

works to set a period as a record separator on UNIX operating systems, but on Windows operating systems it does not. For Windows operating systems, the awk utility requires the following syntax:

```bash
awk -F'.' '{ print $2 }'
```

**Note:** The endpoint label identified in the following example is the lcfd proposed label and is almost always the host name, which could be fully qualified (spot.dev.tivoli.com) or not (spot). You can change the label with the `-Dlcs.machine_name=name` option. See the lcfd command for more information.

```bash
#!/bin/sh
#
# Please do not remove the below Tivoli comments
# --- Start of Tivoli comments ---
#
# The following are the command line options passed to this script
# from the Endpoint Manager.
#
# $1 - The label of the endpoint machine
# $2 - The object reference of the endpoint machine
# $3 - The architecture type of the endpoint machine
# $4 - The object reference of the gateway that the endpoint
#      logged into
# $5 - The ip/ipx address of the endpoint logging in (refer to
#      parameter $10 to determine the protocol of the endpoint).
```
allow_install_policy

# $6 - region
# $7 - dispatcher
# $8 - version# $9 - The inventory id of the endpoint logging in.
# $10 - The protocol of the endpoint logging in.
TCP/IP -> TCP/IP
IPX    -> IPX/SPX

# A line with this format may be written to standard output to
# change an endpoint label:
# new_label = <label>

# The normal exit code of 0 from the allow_install_policy will allow
# the endpoint initial login to proceed. (If the label of this
# endpoint is in use, or invalid, then this login will not complete.)
# An exit code of 10 also will allow this login to proceed and,
# if this endpoints label matches the label of an existing
# endpoint, or if the endpoint label is invalid, then a unique
# label will be created for this endpoint.
# An exit code of 6 will cause this login to be ignored.
#
# Exiting the allow_install_policy with any other non-zero exit status will
# stop this endpoint initial (or orphaned) login.
# The environment variable LCF_LOGIN_STATUS is also set by the epmgr.
# A value of 2 indicates the endpoint is isolated. That is, it was unable
# to contact its assigned gateway. Isolated endpoints are automatically
# migrated to another gateway unless the select_gateway_policy terminates
# with a non-zero exit status. Other LCF_LOGIN_STATUS values are:
# 0  Initial login    (allow_install_policy, select_gateway_policy,
#                      after_install_policy)
# 2  Isolated login  (select_gateway_policy)
# 3  Migratory login (select_gateway_policy)
# 7  Orphaned login  (allow_install_policy, select_gateway_policy,
#                      after_install_policy)
# The allow_install_policy will have these environment variables set if
# there is already an existing endpoint with the same label as the endpoint
# which is attempting to login:
# LCF_DUPL_OBJECT   object id of existing endpoint
# LCF_DUPL_ADDRESS  network address of existing endpoint
# LCF_DUPL_LOGIN    timestamp of existing endpoint first normal login
# LCF_DUPL_GATEWAY  object id of existing endpoint gateway
# LCF_DUPL_INV_ID   inventory id of existing endpoint
# LCF_DUPL_INTERP   interp (architecture type) of existing endpoint
# The initial login will fail for an endpoint whose label matches the label
# of an existing endpoint, unless allow_install_policy is exited with code 10.
# The allow_install_policy will have the environment variable
# LCF_INVALID_LABEL set to TRUE, if the endpoint label, $1, is invalid.
# Endpoint labels must not contain any invalid characters and must conform
# to the labelspace regular expression. A label may be tested with the
# wepmgr test label command. Invalid characters and the labelspace regular
# expression may be displayed and set with the wepmgr get and set commands.
# An initial login will fail if the endpoint label is invalid, unless
# allow login policy is exited with code 10, in which case a generic
# label stem, "eplabel", is used as the beginning of the endpoint label.
# The object dispatcher number and arbitrary characters will be added to
# make the label unique.
# Also note that during the execution of allow_install and select_gateway
# policy scripts, the endpoint does not yet formally exist. For this reason,
# the endpoint object reference will have a value of OBJECT_NIL and the
# object dispatcher number will be 0. The endpoint label will have the value
# suggested by the endpoint (or the user value lcfd -n) but is not guaranteed
# to become the final endpoint label. It will become the final endpoint label
# if this value is not already taken by another endpoint.
# --- End of Tivoli comments ---
# set -e

# Don't allow endpoints from subnet 26 log into this TMR.
SUBNET=`echo $5 | awk '{FS=","} print $1","$2","$3``
if [ "$SUBNET" = "146.84.26" ]; then
  exit 1
fi

# Don't allow endpoints whose name contain the regular
# expression "dev".
# This line will force the script to exit nonzero if the
# expression "dev" is in the label.
# echo $1 | grep -v dev

exit 0
after_install_policy

Performs any processing that you want immediately after the endpoint has been created.

Syntax

after_install_policy endpoint_label endpoint_object_id endpoint_interpreter_type gateway_object_id endpoint_ip_address region dispatcher version unique_id protocol

Description

The after_install_policy script performs any processing that you want immediately after the endpoint is created. This policy is run only after the initial login; it is not run on subsequent logins.

The failure of this script does not stop the login process. The endpoint already exists when this script is run. Only the post-processing specified in the script fails.

The endpoint manager runs after_install_policy immediately following the initial gateway assignment but before the endpoint login method returns to the intercepting gateway. Because this occurs before the first normal login for an endpoint, you cannot run downcalls from this script.

Options

dispatcher
    Specifies the object dispatcher number of the endpoint.

endpoint_interpreter_type
    Specifies the interpreter type of the endpoint.

endpoint_address
    Specifies the Internet Protocol (IP) address and port number of the endpoint.

    Note: When hostname resolution is enabled using the wepmgr ep_hostname_resolution command, this value can be either a hostname or an IP address.

endpoint_label
    Specifies the label of the endpoint for which the script will run.

endpoint_object_id
    Specifies the object ID of the endpoint.

gateway_object_id
    Specifies the object ID of the assigned gateway.

protocol
    Specifies the network protocol used by the endpoint.

region
    Specifies the region number in which the endpoint resides.

unique_id
    Specifies the unique ID (also called the inventory ID) of the endpoint.

version
    Specifies the current version of the endpoint software.
Examples

The following example subscribes new endpoints to a profile manager that represents endpoints of similar architecture types. If the policy region or profile manager does not exist, the policy creates it.

```bash
#!/bin/sh
# Please do not remove the below Tivoli comments
# --- Start of Tivoli comments ---
#
# The following are the command line options passed to
# this script from the Endpoint Manager.
#
# $1 - The label of the endpoint machine
# $2 - The object reference of the endpoint machine
# $3 - The interpreter type of the endpoint machine
# $4 - The object reference of the assigned gateway that the
#     endpoint logged into
# $5 - The IP address of the endpoint logging in
# $6 - Region
# $7 - Dispatcher
# $8 - Version
# $9 - The inventory id of the endpoint logging in
#$10 - The protocol of the endpoint logging in.
# TCP/IP -> TCP/IP
# IPX -> IPX/SPX
#
# The environment variable LCF>Login_STATUS is also set by the epmgr.
# A value of 2 indicates the endpoint is isolated. That is, it was unable
# to contact its assigned gateway. Isolated endpoints are automatically
# migrated to another gateway unless the select_gateway_policy terminates
# with a non-zero exit status. Other LCF_LOGIN_STATUS values are:
# 0 Initial login (allow_install_policy, select_gateway_policy,
#    after_install_policy)
# 2 Isolated login (select_gateway_policy)
# 3 Migratory login (select_gateway_policy)
# 7 Orphaned login (allow_install_policy, select_gateway_policy,
#    after_install_policy)
# --- End of Tivoli comments ---
LCF_POLICY_REGION=LCF-Endpoints
PROFILE_MANAGER=LCF-$3
EP=$1

# Check to see if our top-level policy region already
# exists. If not create it and put it on this administrators
# desktop.
#
# Disable "exit on error" for this call since we will handle
# the failure.
set +e
wlookup -r PolicyRegion $LCF_POLICY_REGION > /dev/null
ERR=$?
set -e
if [ $ERR -ne 0 ]; then
    ALI="objcall 0.0.0 get_security_objid"
    set "objcall $ALI get_identity"
    ADMIN="$1"
    ADMIN_OID=`echo $2 |cut -d"#" -f1`
    wcrtpr -m ProfileManager -a $ADMIN $LCF_POLICY_REGION
    idlcall $ADMIN_OID refresh_collection
fi

# Check to see if our interp specific profile manager
```
after_install_policy

# already exists. If not create it and make it dataless so
# that we can subscribe the endpoint to it.
#
# Disable "exit on error" for this call since we will handle
# the failure.
#
set +e
wlookup -r ProfileManager $PROFILE_MANAGER > /dev/null
ERR=$?
set -e

if [ $ERR -ne 0 ]; then
    wcrtprfmgr $LCF_POLICY_REGION $PROFILE_MANAGER > /dev/null
    wsetpm -d /Library/ProfileManager/$PROFILE_MANAGER
fi

#
# Subscribe the endpoint to the profile manager which
# contains the endpoints for that specific interp type.
#
wsub /Library/ProfileManager/$PROFILE_MANAGER \
@Endpoint:$EP

exit 0
login_policy

Performs any processing that you want each time an endpoint logs in.

Syntax

```
login_policy endpoint_label endpoint_object_id endpoint_interpreter_type
gateway_object_id endpoint_ip_address region dispatcher version unique_id protocol
```

Description

The `login_policy` script performs any processing that you want each time an endpoint logs in. This policy is run by the gateway to which the endpoint is assigned. The same policy script is run on all the gateways in a Tivoli region.

Note: This policy does not support the use of binaries.

One feature of this script is that it can be configured to automatically upgrade the endpoint software on each client. To configure the `login_policy` script to upgrade endpoint software, follow these steps:

1. Enable the upgrade script (`upgrade.sh`) in `BINDIR/../lcf_bundle/upgrade` by changing the `upgrade_mode` entry to `auto` in the `upgrade.cntl` file. This step must be performed on a per gateway basis.
2. Edit the `login_policy` script to call `upgrade.sh`. The following example script includes this option. The upgrade script does not log output unless defined to do so.

The endpoint gateway runs `login_policy` each time an endpoint logs in.

Options

- `dispatcher`
  Specifies the object dispatcher number of the endpoint.
- `endpoint_interpreter_type`
  Specifies the interpreter type of the endpoint.
- `endpoint_address`
  Specifies the Internet Protocol (IP) address and port number of the endpoint.
- `endpoint_label`
  Specifies the label of the endpoint for which the script will run.
- `endpoint_object_id`
  Specifies the object ID of the endpoint.
- `gateway_object_id`
  Specifies the object ID of the assigned gateway.
- `protocol`
  Specifies the network protocol used by the endpoint.
- `region`
  Specifies the region number in which the endpoint resides.
- `unique_id`
  Specifies the unique ID (also called the inventory ID) of the endpoint.
- `version`
  Specifies the current version of the endpoint software.
Examples

The following example logs a notice to an endpoint-related notice group every time an endpoint logs in and automatically upgrades the endpoint software:

```bash
#!/bin/sh

# Please do not remove the below Tivoli comments
# --- Start of Tivoli comments ---
#
# The following are the command line options passed to this script
# from the Gateway.
#
# $1 - The label of the endpoint machine
# $2 - The object reference of the endpoint machine
# $3 - The architecture type of the endpoint machine
# $4 - The object reference of the gateway that the endpoint logged in
# $5 - The ip/ipx address of the endpoint logging in (refer to parameter
#   $10 to determine the protocol of the endpoint).
# $6 - region
# $7 - dispatcher
# $8 - version
# $9 - The inventory id of the endpoint
# $10 - The protocol of the endpoint logging in.
#
# --- End of Tivoli comments ---
#
# AUTO UPGRADE
# Invoke the upgrade script to check the current version of
# the endpoint software and upgrade if necessary.
B0="objcall 0.0.0 self"
OS="objcall $B0 getattr oserv"
INSTALLDIR="objcall $OS query install_dir|tr '\134' '/'"
$INSTALLDIR/lcf_bundle/upgrade/upgrade.sh $1 $8 $3

# LCF_NOTICE_GROUP=LCF_Endpoints
#
# Send a notice to LCF endpoint notice group every time this
# endpoint logs in.
#
# set +e
wlookup -r TMF_Notice $LCF_NOTICE_GROUP > /dev/null
ERR=$?
set -e
if [ $ERR -ne 0 ]; then
  NTFGM="wlookup -r Classes TMF_Notice"
  idlcall -T top $NTFGM \n    TMF_Notice::NoticeManager::create_notice_group \n    "'$LCF_NOTICE_GROUP'" 72'
fi

GW="idlcall $4 _get_label"
EPOID="wlookup -o -r Endpoint $1"

wsndnotif $LCF_NOTICE_GROUP Notice << LCF_NOTICE
Endpoint $1 ($EPOID of interp type, $3, logged into gateway
$GW ($4).
LCF_NOTICE
exit 0
```
select_gateway_policy

Determines the set of gateways that are allowed to manage the endpoint.

Syntax

```
select_gateway_policy endpoint_label endpoint_object_id endpoint_interpreter_type
gateway_object_id endpoint_ip_address region dispatcher version unique_id protocol
```

Description

The `select_gateway_policy` script determines the set of gateways that are allowed to manage the endpoint. `select_gateway_policy` is run at initial login, when an endpoint is isolated, or when an endpoint performs a migratory login.

If the script does not return gateways, the endpoint manager performs its default selection process and generates a list of up to five gateways in the endpoint’s region. The endpoint manager’s default selection process is overridden by `select_gateway_policy`. If this policy is not defined, the intercepting gateway is added to the end of the list of candidate gateways.

The intercepting gateway is also added to the end of the `select_gateway_policy` list to ensure that the endpoint has at least one definite contact. If the endpoint manager cannot contact any of the gateways listed in the script, the endpoint manager assigns the intercepting gateway to the endpoint. If the script fails (returns nonzero), the login attempt fails. If `select_gateway_policy` runs successfully, the object ID of the assigned gateway is passed to the endpoint.

The endpoint manager runs `select_gateway_policy` when it receives the initial login packet for the endpoint after the `allow_install_policy` is run.

Options

- `dispatcher`
  Specifies the object dispatcher number of the endpoint.
- `endpoint_interpreter_type`
  Specifies the interpreter type of the endpoint.
- `endpoint_address`
  Specifies the Internet Protocol (IP) address and port number of the endpoint.

  **Note:** When hostname resolution is enabled using the `wepmgr ep_hostname_resolution` command, this value can be either a hostname or an IP address.
- `endpoint_label`
  Specifies the label of the endpoint for which the script will run.
- `endpoint_object_id`
  Specifies the object ID of the endpoint.
- `gateway_object_id`
  Specifies the object ID of the assigned gateway.
- `protocol`
  Specifies the network protocol used by the endpoint.
- `region`
  Specifies the policy region in which the endpoint resides.
select_gateway_policy

unique_id
   Specifies the unique ID (also called the inventory ID) of the endpoint.

version   Specifies the current version of the endpoint.

Examples
The options passed to this script from the endpoint manager are the same as the
previous three endpoint policy scripts. The LCF_LOGIN_STATUS variable is also
set by the endpoint manager. A value of 2 indicates that the endpoint is isolated
(unable to contact its assigned gateway). Isolated endpoints are automatically
migrated to another gateway unless the select_gateway_policy script terminates
with a nonzero exit status. For more information about endpoint isolation or
migration, see the Tivoli Management Framework Planning for Deployment Guide.

Note: During the execution of this script, the endpoint does not yet formally exist.
Therefore, the value of the endpoint object reference is OBJECT_NIL and the
object dispatcher number is 0. The value of the endpoint label is suggested
by the endpoint (or the user through the -n option of the lcfd command),
but can only become the final label if the value is not already taken by
another endpoint.

If your gateways and endpoints are separated by a network address translation
(NAT) device, you must use the fully qualified host name of the gateway
appended to its object identifier with a pipe ( | ) symbol. For example, a gateway
paris fully qualified as paris.dev.server.com with an object identifier of
123267682.1.529 should be entered in the select_gateway_policy script as follows:
123267682.1.529|paris.dev.server.com

The following is an example script:
#!/bin/sh
# Please do not remove the below Tivoli comments
# --- Start of Tivoli comments ---
#
# The following are the command line options passed to
# this script from the Endpoint Manager.
#
# # $1 - The label of the endpoint machine
# $2 - The object reference of the endpoint machine
# $3 - The interpreter type of the endpoint machine
# $4 - The object reference of the assigned gateway that the
# endpoint logged into
# $5 - The IP address of the endpoint logging in
# $6 - Region
# $7 - Dispatcher
# $8 - Version
# $9 - The inventory id of the endpoint logging in
# $10 - The protocol of the endpoint logging in.
# TCPIP -> TCP/IP
# IPX -> IPX/SPX
#
# The environment variable LCF_LOGIN_STATUS is also set by the epmgr.
# A value of 2 indicates the endpoint is isolated. That is, it was unable
# to contact its assigned gateway. Isolated endpoints are automatically
# migrated to another gateway unless the select_gateway_policy terminates
# with a non-zero exit status. Other LCF_LOGIN_STATUS values are:
# 0 Initial login  (allow_install_policy, select_gateway_policy,
#                  after_install_policy)
# 2 Isolated login (select_gateway_policy)
# 3 Migratory login (select_gateway_policy)
# 7 Orphaned login (allow_install_policy, select_gateway_policy,
#                  after_install_policy)
Also note that during the execution of allow_install and select_gateway policy scripts, the endpoint does not yet formally exist. For this reason, the endpoint object reference will have a value of OBJECT_NIL and the object dispatcher number will be 0. The endpoint label will have the value suggested by the endpoint (or the user value lcfd -n) but is not guaranteed to become the final endpoint label. If this value is not already taken by another endpoint.

NB: That the version in $8 is not defined when the sgp is called during a migratory completion by login or upcall.

--- End of Tivoli comments ---

# only ep_ip is needed for this example
ep_label=$1
ep_oid=$2
ep_interp=$3
gateway=$4
ep_ip=$5
region=$6
dispatcher=$7
version=$8

# FOUNDONE=FALSE
# we just want the subnet of the endpoint
SUBNET=`echo $ep_ip|cut -d'.' -f3`
# get all gateways and find ones that are on the same subnet
GATEWAYS=`wlookup -ar Gateway -o`

for gwoid in $GATEWAYS
    do
gwproxy=`idlattr -tg $gwoid proxy Object`
    mnips=`wifconfig -h $gwproxy | grep -v Device | awk '{print $2}"
    # a managed node might have multiple interfaces, so check each
    # one of them if the gateway subnet matches the endpoint number.
    # return gwoid if it matches
    for ip in $mnips
        do
gwsub=`echo $ip | cut -d'.' -f3`
        if [ $gwsub -eq $SUBNET ]; then
echo $gwoid
    FOUNDONE=TRUE
    fi
done
    done

# if you did not find a gateway, and you still want the endpoint
# to log in, exit 0, else exit 1
if [ "$FOUNDONE" = "TRUE" ]; then
    exit 1
else
    exit 0
fi
Chapter 5. Profile manager policy scripts

Tivoli Management Framework provides default and validation policy for the profile manager service.

The profile manager default policy defines which managed nodes can become subscribers to a profile manager and defines the profile managers into which a profile can be cloned.

The validation policy validates the following:
- Whether a subscriber can be removed from a profile manager
- Whether a profile manager can cancel a subscription
- Whether a subscriber can be added to a profile manager
- Whether a profile manager can subscribe to another profile manager

Default policy methods for profile managers

The following table contains the profile manager default policy methods and their purpose.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_def_profile_managers</td>
<td>Provides a list of profile managers into which a profile can be cloned.</td>
</tr>
<tr>
<td>pm_def_profile_types</td>
<td>Provides a list of profile types managed by a given region.</td>
</tr>
<tr>
<td>pm_def_subscribers</td>
<td>Provides a list of managed nodes that can become subscribers to a profile manager.</td>
</tr>
</tbody>
</table>

Validation policy methods for profile managers

The following table contains the profile manager validation policy methods and their purpose.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_val_remove_subscribers</td>
<td>Validates the removal of subscribers from a profile manager.</td>
</tr>
<tr>
<td>pm_val_remove_subscription</td>
<td>Validates the cancellation of a subscription to a profile manager.</td>
</tr>
<tr>
<td>pm_val_subscribers</td>
<td>Validates the addition of subscribers to a profile manager.</td>
</tr>
<tr>
<td>pm_val_subscription</td>
<td>Validates the subscription of a profile manager to another profile manager.</td>
</tr>
</tbody>
</table>

Profile manager policy methods

This section describes the default and validation policy methods for profile managers.
**pm_def_profile_managers**

Provides a list of profile managers into which a profile can be cloned.

**Resource**

ProfileManager

**Syntax**

```
pm_def_profile_managers manager_name
```

**Description**

The `pm_def_profile_managers` method provides a list of profile managers into which a profile can be cloned.

The method runs the `pm_def_profile_managers` script. The script writes the profile manager list to standard output, one profile manager per line. This script can be customized. The output of the script must be in the format `label`, followed by the tab character, followed by the object ID.

For example, if the label is PM1 and the OID is 1214115201.1.616#TMF_CCMS::ProfileManager#, the following ASCII characters must be written to standard output:

```
PM1	1214115201.1.616#TMF_CCMS::ProfileManager#
```

The output is displayed in the **Clone to Profile Managers** scrolling list in the Clone Profile window.

The default implementation of `pm_def_profile_managers` returns a list of all instances in all Tivoli management regions of the ProfileManager resource type.

**Options**

```
manager_name
```

Specifies the name of the profile manager containing the profile to clone.

**Return Codes**

This method exits with one of the following:

- **0** Indicates the successful completion of the method.
- **1** Indicates that the method exited with an error. The standard output is undefined.
pm_def_profile_types

Provides a list of profile types managed by a given region.

Resource
ProfileManager

Syntax

pm_def_profile_types region

Description
The pm_def_profile_types method provides a list of profile types managed by a given region.

The method runs the pm_def_profile_types script. The script writes the profile types list to standard output, one profile type per line. The output is displayed in the Create Profile window.

The default implementation of pm_def_profile_types returns a list of all profile types managed by the given region.

Options

region Specifies the name of the policy region in which a profile is to be created.

Return Codes

This method exits with one of the following:

0 Indicates the successful completion of the method; the method writes profile types to standard output.

1 Indicates that the method exited with an error. The standard output is undefined.
**pm_def_subscribers**

Provides a list of managed nodes that can become subscribers to a profile manager.

**Resource**

ProfileManager

**Syntax**

```
pm_def_subscribers manager_name
```

**Description**

The **pm_def_subscribers** method provides a list of managed nodes, profile managers, and endpoints that can become subscribers to a profile manager.

The method runs the **pm_def_subscribers** script. The script writes the subscriber list to standard output, one subscriber per line. This script can be customized. The output of the script must be in the format `label`, followed by the tab character, followed by the object ID.

For example, if the label is reality and the OID is 1214115201.3.7#TMF_ManagedNode::ManagedNode#, the following ASCII characters must be written to standard output:

```
reality\t1214115201.3.7#TMF_ManagedNode::ManagedNode#
```

The output is displayed in the **Available to become Subscribers** scrolling list in the Subscriber window.

The default implementation of **pm_def_subscribers** returns a list of all instances in all Tivoli regions of the ProfileManager and ManagedNode resource types.

**Options**

```
manager_name
```

Specifies the name of the profile manager that is to be subscribed to.

**Return Codes**

This method exits with one of the following:

0 Indicates the successful completion of the method.

1 Indicates that the method exited with an error. The standard output is undefined.

**See Also**

[pm_val_remove_subscribers](#), [pm_val_remove_subscription](#), [pm_val_subscribers](#), [pm_val_subscription](#)
pm_val_remove_subscribers

Validates the removal of subscribers from a profile manager.

**Resource**
ProfileManager

**Syntax**

```
pm_val_remove_subscribers localize manager_name subscriber...
```

```
pm_val_remove_subscribers delete manager_name subscriber...
```

**Description**
The `pm_val_remove_subscribers` method validates the removal of subscribers from a profile manager.

The method runs the `pm_val_remove_subscribers` script. The script writes TRUE to standard output if the removal meets the validation criteria, FALSE if it does not.

Tivoli Management Framework provides the default value of TRUE.

**Options**

- `delete`  Indicates that this action will be taken when removing the subscribers.
- `localize` Indicates that this action will be taken when removing the subscribers.

```
manager_name
```
Specifies the name of the profile manager.

```
subscriber...
```
Specifies the list of subscribers being removed. Separate each subscriber with a space.

**Return Codes**

This method exits with one of the following:

- **0**  Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.
- **1**  Indicates that the method exited with an error. The standard output is undefined.

**See Also**

- `pm_val_remove_subscription`
- `pm_val_subscription`
**pm_val_remove_subscription**

Validates the cancellation of a subscription to a profile manager.

**Resource**

ProfileManager

**Syntax**

```
pm_val_remove_subscription subscriber localize manager_name

pm_val_remove_subscription subscriber delete manager_name
```

**Description**

The `pm_val_remove_subscription` method validates the cancellation of a subscription to a profile manager by a profile manager or endpoint. The method runs the `pm_val_remove_subscription` script. The script writes TRUE to standard output if the removal meets the validation criteria, FALSE if it does not.

Tivoli Management Framework provides the default value of TRUE.

**Options**

- **delete** Indicates that this action will be taken when removing the subscribers.
- **localize** Indicates that this action will be taken when removing the subscribers.
- **manager_name** Specifies the profile manager to which the subscriber subscribes.
- **subscriber** Specifies the name of the profile manager or endpoint that is the subscriber.

**Return Codes**

This method exits with one of the following:

- **0** Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.
- **1** Indicates that the method exited with an error. The standard output is undefined.

**See Also**

- `pm_val_remove_subscribers`
- `pm_val_subscription`
pm_val_subscribers

Validates the addition of subscribers to a profile manager.

Resource

ProfileManager

Syntax

pm_val_subscribers manager_name subscriber...

Description

The pm_val_subscribers method validates the addition of subscribers to a profile manager. The method runs the pm_val_subscribers script. The script writes TRUE to standard output if the new subscribers meet the validation criteria, FALSE if they do not.

Tivoli Management Framework provides the default value of TRUE.

Options

manager_name

Specifies the name of the profile manager.

subscriber...

Specifies the list of new subscribers to validate. Separate each subscriber with a space.

Return Codes

This method exits with one of the following:

0 Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.

1 Indicates that the method exited with an error. The standard output is undefined.

See Also

pm_val_remove_subscribers  pm_val_remove_subscription  pm_val_subscription
pm_val_subscription

Validates the addition of a subscription of a profile manager to another profile manager.

Resource
    ProfileManager

Syntax

    pm_val_subscription subscribee manager_name

Description

The pm_val_subscription method validates the addition of a subscription to a profile manager by a profile manager or endpoint. The method runs the pm_val_subscription script. The script writes TRUE to standard output if the subscription meets the validation criteria, FALSE if it does not.

Tivoli Management Framework provides the default value of TRUE.

Options

    manager_name
        Specifies the profile manager or endpoint that is the subscriber.

    subscribee
        Specifies the profile manager that is the subscribee.

Return Codes

This method exits with one of the following:

    0    Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.

    1    Indicates that the method exited with an error. The standard output is undefined.

See Also

    pm_val_remove_subscribers pm_val_remove_subscription
Chapter 6. Task library policy scripts

Within the task library, default policy sets the list of managed nodes and profiles managers that are associated with a task or job when it is created.

Validation policy verifies the following:

- That existing tasks or jobs are associated with the correct managed nodes and profile managers
- That the effective group and user IDs under which the task or job will run are valid

Default policy methods for task libraries

The following table contains the task library default policy methods and their purpose.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tl_def_dist_mode</td>
<td>Provides the default mode for distributing task binaries throughout a Tivoli region.</td>
</tr>
<tr>
<td>tl_def_man_nodes</td>
<td>Provides a default list of managed nodes and endpoints for a task library.</td>
</tr>
<tr>
<td>tl_def_prof_mgrs</td>
<td>Provides a default list of profile managers for a task library.</td>
</tr>
<tr>
<td>tl_def_set_gid</td>
<td>Provides the default group ID assigned to a task.</td>
</tr>
<tr>
<td>tl_def_set_uid</td>
<td>Provides the default user ID assigned to a task.</td>
</tr>
</tbody>
</table>

Validation policy methods for task libraries

The following table contains the task library validation policy methods and their purpose.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tl_val_man_nodes</td>
<td>Validates the list of target managed nodes and endpoints on which a task or job will execute.</td>
</tr>
<tr>
<td>tl_val_prof_mgrs</td>
<td>Validates the list of target profile managers associated with a task or job.</td>
</tr>
<tr>
<td>tl_val_set_gid</td>
<td>Validates the effective group ID assigned to a task or job.</td>
</tr>
<tr>
<td>tl_val_set_uid</td>
<td>Validates the effective user ID assigned to a task or job.</td>
</tr>
</tbody>
</table>

Task library policy methods

This section describes the default and validation policy methods for task libraries.
tl_def_dist_mode

Provides the default mode for distributing task binaries throughout a Tivoli region.

Resource
TaskLibrary

Syntax
	tl_def_dist_mode

Description
The `tl_def_dist_mode` method provides the default mode used to distribute task executable files when a task is created. The following are valid distribution modes:

- **ALI**: Copies task binaries to the Tivoli server only.
- **GLOBAL**: Copies task binaries to all managed nodes in all connected Tivoli regions.
- **LOCAL**: Copies task binaries to all managed nodes in the local Tivoli region.

Return Codes
This method exits with one of the following:

- **0**: Indicates the successful completion of the method.
- **1**: Indicates that the method was invoked with invalid options.
- **2**: Indicates an error.
**tl_def_man_nodes**

Provides a default list of managed nodes and endpoints for a task library.

**Resource**

TaskLibrary

**Syntax**

```
tl_def_man_nodes task_name administrator_name
```

**Description**

The `tl_def_man_nodes` method provides a default list of managed nodes and endpoints for a task library. The `tl_def_man_nodes` method runs the `tl_def_man_nodes.sh` script. This script must provide a list of ManagedNode and Endpoint resources where the task can run.

The method writes the list of managed nodes and endpoints to standard output, one node per line.

**Options**

- `administrator_name`
  - Specifies the name of the administrator that invoked the method.
- `task_name`
  - Specifies the task for which the list is being generated.

**Return Codes**

This method exits with one of the following:

- 0 Indicates the successful completion of the method.
- 1 Indicates that the method was invoked with invalid options.
- 2 Indicates an error.

**See Also**

[tl_def_prof_mgrs](#)
**tl_def_prof_mgrs**

Provides a default list of profile managers for a task library.

**Resource**

   TaskLibrary

**Syntax**

    tl_def_prof_mgrs task_name administrator_name

**Description**

The `tl_def_prof_mgrs` method provides a default list of profile managers for a task library. This script must provide a list of ProfileManager resources where the task can run.

The method writes the list of profile managers to standard output, one per line.

**Options**

   administrator_name
       Specifies the name of the administrator that invoked the method.

   task_name
       Specifies the task for which the list is being generated.

**Return Codes**

This method exits with one of the following:

0   Indicates the successful completion of the method.
1   Indicates that the method was invoked with invalid options.
2   Indicates an error.

**See Also**

   [tl_val_man_nodes](#)
**tl_def_set_gid**

Provides the default group ID assigned to a task.

**Resource**
TaskLibrary

**Syntax**

```
 tl_def_set_gid administrator_name
```

**Description**

The `tl_def_set_gid` method provides the default group ID associated with a task.

The script writes the group ID to standard output. The output is displayed in the Create Task window.

**Options**

```
 administrator_name
```

Specifies the name of the administrator that invoked the method.

**Return Codes**

This method exits with one of the following:

- **0** Indicates the successful completion of the method; the method writes the group ID to standard output.

- **1** Indicates that the method exited with an error. The method’s standard output is undefined.

**See Also**

[tl_def_set_uid](#)
tl_def_set_uid

Provides the default user ID assigned to a task.

Resource

TaskLibrary

Syntax

\texttt{tl\_def\_set\_uid} \texttt{administrator\_name}

Description

The \texttt{tl\_def\_set\_uid} method provides the default user ID assigned to a task.

The script writes the user ID to standard output. The output is displayed in the Create Task window.

Options

\texttt{administrator\_name}

Specifies the name of the administrator that invoked the method.

Return Codes

This method exits with one of the following:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates the successful completion of the method; the method writes the user ID to standard output.</td>
</tr>
<tr>
<td>1</td>
<td>Indicates that the method exited with an error. The method’s standard output is undefined.</td>
</tr>
</tbody>
</table>

See Also

\texttt{allow\_install\_policy}
tl_val_man_nodes

Validates the list of target managed nodes and endpoints associated with a task or job.

Resource
TaskLibrary

Syntax
tl_val_man_nodes task_name administrator_name node_name...

Description
The tl_val_man_nodes method validates the list of managed nodes and endpoints associated with a task or job.

The script writes TRUE to standard output if the list meets the validation criteria, or writes FALSE if it does not. Tivoli Management Framework provides the default value of TRUE.

Options

administrator_name
Specifies the name of the administrator that invoked the method.

node_name
Specifies the list of managed nodes and endpoints to validate. Separate each name with a space.

task_name
Specifies the name of the task or job.

Return Codes
This method exits with one of the following:

0 Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.

1 Indicates that the method exited with an error. The standard output for the method is undefined.

See Also
[tl_def_prof_mgrs tl_val_set_gid tl_val_set_uid]
**tl_val_prof_mgrs**

Validates the list of target profile managers associated with a task or job.

**Resource**

TaskLibrary

**Syntax**

```
tl_val_prof_mgrs task_name administrator_name profile_manager_name...
```

**Description**

The `tl_val_prof_mgrs` method validates the list of target profile managers associated with a task or job.

The script writes TRUE to standard output if the list meets the validation criteria, FALSE if it does not. The default value is TRUE.

**Options**

- `administrator_name`
  Specifies the name of the administrator that invoked the method.

- `profile_manager_name...`
  Specifies the list of profile managers to validate. Separate each name with a space.

- `task_name`
  Specifies the name of the task or job.

**Return Codes**

This method exits with one of the following:

- **0** Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.

- **1** Indicates that the method exited with an error. The method’s standard output is undefined.

**See Also**

- `tl_def_man_nodes`
- `tl_val_set_gid`
- `tl_val_set_uid`
tl_val_set_gid

Validates the group ID assigned to a task or job.

Resource
TaskLibrary

Syntax

tl_val_set_gid administrator_name group_id

Description
The tl_val_set_gid method validates the group ID associated with a task or job. The script writes TRUE to standard output if the group ID meets the validation criteria, FALSE if it does not. Tivoli Management Framework provides the default value of TRUE.

Options

    administrator_name
        Specifies the name of the administrator that invoked the method.

    group_id
        Specifies the group ID to validate.

Return Codes
This method exits with one of the following:

    0  Indicates the successful completion of the method; the method writes either TRUE or FALSE to standard output.

    1  Indicates that the method exited with an error. The standard output is undefined.

See Also

tl_def_man_nodes tl_val_set_uid
tl_val_set_uid

Validates the user ID assigned to a task or job.

Resource
TaskLibrary

Syntax

```
    tl_val_set_uid administrator_name user_id
```

Description

The `tl_val_set_uid` method validates the user ID associated with a task or job. The script writes `TRUE` to standard output if the user ID meets the validation criteria, `FALSE` if it does not. Tivoli Management Framework provides the default value of `TRUE`.

Options

```
    administrator_name
        Specifies the name of the administrator that invoked the method.

    user_id
        Specifies the user ID to validate.
```

Return Codes

This method exits with one of the following:

```
    0  Indicates the successful completion of the method; the method writes
        either `TRUE` or `FALSE` to standard output.

    1  Indicates that the method exited with an error. The standard output is
        undefined.
```

See Also

```
    tl_def_man_nodes  tl_val_prof_mgrs  tl_val_set_gid
```
Chapter 7. Editing policy scripts

Editing endpoint policy scripts

Tivoli Management Framework is installed with empty endpoint policy scripts. To add content to these scripts or, later, to edit existing scripts, you must use the **wgeteppol** and **wputeppol** commands. Refer to Part 1, “Platform commands,” on page 1 for more information.

Use the following steps to edit an endpoint policy script. This example uses the **login_policy** script. The same procedure works for all endpoint policy scripts by replacing **login_policy** with the name of the policy you want to edit.

**Note:** If you do not redirect the policy to a file, the policy is written to your screen.

1. Enter the following command to extract **login_policy** and write it to a file:

   ```
   wgeteppol login_policy > login_policy.txt
   ```

   If you are editing this script for the first time, you see the following output. On subsequent edits, you see the entire script.

   ```
   #!/bin/sh
   #
   # The following are the command line options passed
   # to this script from the gateway.
   #
   # $1 - The label of the endpoint machine
   # $2 - The object reference of the endpoint machine
   # $3 - The interpreter type of the endpoint machine
   # $4 - The object reference of the gateway that the
   #      endpoint logged into
   # $5 - The IP address of the endpoint logging in
   # $6 - Region
   # $7 - Dispatcher
   # $8 - Version
   # $9 - The inventory id of the endpoint logging in
   # $10 - The protocol of the endpoint logging in.
   # TCP/IP -> TCP/IP
   # IPX -> IPX/SPX
   
   exit 0
   #
   ```

2. Use a text editor to add the contents of the script or to modify existing content.

3. Enter the following command to return the updated policy script:

   ```
   wputeppol login_policy < login_policy.txt
   ```

Editing profile manager and task library policy scripts

The following example demonstrates the process for editing task library validation policy. You can use the same procedures for editing profile manager validation policy. To edit default policy, replace the **–v** option with the **–d** option on the **wlspolm**, **wgetpolm**, and **wputpolm** commands.
As shipped, the task library policy sets the valid managed nodes on which a job or task can be run to all managed nodes in the Tivoli region. By modifying the validation policy, you can limit the managed nodes on which the job or task actually executes.

1. Enter the following command to return the list of available validation policies and their proper names:
   
   `wlspol -v TaskLibrary`

2. Enter the following command to get the policy shell script associated with the `validate_execution_managed_nodes` method and redirect it to the `aef` file. You must use the proper name.
   
   `wgetpolm -v TaskLibrary BasicTaskLibrary tl_val_man_nodes > aef`

   The `aef` file contains a shell script similar to the following:
   
   ```
   #!/bin/sh
   # This script implements the
   # "validate_execution Managed_nodes" policy method for the
   # Task Library. The script is provided with the name of the
   # task, the label of the Admin and all of the
   # managed nodes selected for execution targets of the task.
   # Modify the code below if you want something different
   # returned.
   # To debug your changes, you could add the lines:
   #
   # set -xv
   # exec > /tmp/debug.output 2>&1
   #
   # These lines will allow you to see any errors that occur
   # by looking in the /tmp/debug.output file.
   #
   # NOTE: This script can also be called when a check_policy
   # operation is performed. In that case, the name of
   # the Admin will be "any". Make sure that you handle
   # that case if you modify this script.
   #
   task_name=$1
   administrator=$2
   shift 2
   
   # Example of how to validate the list of managed nodes. ##
   
   for i in $*; do
     if [ "$i" = "the evil managed node" ]; then
       echo FALSE
       exit 0
     fi
   done
   
   echo TRUE
   exit 0
   #
   ```

3. Use a text editor to modify the `aef` file.

   Tivoli Management Framework provides a policy option to limit the managed node on which the job or task executes. To use this option, remove the comment lines from the policy option Tivoli Management Framework provides and replace "the evil managed node" with the name of a managed node.

   You can also add your own options.
4. Enter the following command to replace the policy with the edited policy script:
   
wputpolm -d TaskLibrary BasicTaskLibrary tl_val_man_nodes < aef

   Enter the following command to create your own policy group:
   
wcrtpol -d TaskLibrary "Secure Tasks"

   You can then get or set methods on this new group and assign the group to the TaskLibrary resource with the Managed Resource Policies window.

   The methods call shell scripts that you can modify to set a new default policy. You can set default policy on a per policy region basis.
### Part 3. Task library language

**Chapter 8. Task library language overview**  435
- Using the wtll command  435
- Using message catalogs with a task library  436
- Enabling tasks to be run on endpoints  437

**Chapter 9. Syntax for the task library language**  439
- Setting library characteristics  439
- Defining Argument layouts  441
  - Text and CryptoText  441
  - Choice and TextChoice  443
    - Example—Choice layout with the Resource keyword  444
    - Example—TextChoice layout with the File keyword  446
    - Example—TextChoice layout with an embedded Choice List  447
  - ChoiceButton  447
  - Example—Program keyword  448
  - Example—Resource keyword  448
  - RadioButton  449
    - Example—File keyword  450
    - Example—embedded list of radio buttons  450
- Creating task definitions  451
  - Defining a task header  451
  - Defining an Implementation clause  452
  - Defining Argument clauses  454
- Adding comments to a task library  456
- Editing the TLL configuration files  456

**Chapter 10. Sample Task Library**  459
Chapter 8. Task library language overview

A task is an action that is routinely performed on managed nodes and endpoints throughout the network. Each task references a script, command, or other executable file that performs the work. A task also defines the authorization role required to execute the script, and the user or group ID the task will execute as. Example tasks include those that clear printer queues, start system backups, or perform Tivoli-supported operations, such as forwarding an event to an event server.

Tasks are stored in a task library. A task library usually contains tasks that are closely related, such as those that require the same authorization role, or those that help manage a single product or resource.

The task library language (TLL) enables you to construct task libraries and tasks. When you create a task library or task from the Tivoli desktop, the desktop uses the TLL to define the basic properties of a task library. The TLL provides additional features that cannot be implemented from the desktop, such as the ability to create customized windows that contain gadgets such as text boxes, radio buttons and scrolling lists. These gadgets can be used to capture user input and pass arguments to the underlying scripts or programs.

Using the wtl1 command

The wtl1 command is the primary command used to create and edit task libraries. This command runs the TLL translator, which is required to convert a task library description into a CORBA (Common Object Request Broker Architecture) data structure. The TLL translator is run from the command line using the wtl1 command. There is currently no graphical user interface for this translator.

After a task library description is converted to a CORBA data structure and is installed in a Tivoli region, the tasks in the task library are available for use by Tivoli applications. The applications configure and launch the tasks. Different applications can launch configured tasks in different ways.

You can view and edit the contents of a task library by running the following command:

```
wtl1 -F export_file -l library_name
```

This command exports the task library into a tar file. The tar file contains the tll file, which is the file that you can edit and re-import. The contents of the task library can be extracted by using the following shell command:

```
tar -xvf file_name
```

The extracted tll file can be edited with any text editor. After you edit the file, use the wtl1 command to import the task library back into the Tivoli region.

**Note:** Importing a task library deletes the previous version of the task library and replaces it with the newly-generated task library. As a result, all jobs and any tasks that were not defined when you exported the library, will be lost.

The following command imports a task library:
where:

–r Replaces the task library. If you renamed the task library when you edited the tll file, omit this flag.

–p policy_region
  Specifies the policy region in which to place the task library.

–P C_preprocessor
  Specifies the path to the C preprocessor. On most UNIX operating systems, this is /usr/ccs/lib/cpp. On Windows managed nodes, the path to the C preprocessor is $BINDIR/tools/cpp.exe.

tll Indicates the file containing the task library definition.

You should maintain a script that can quickly recreate the tasks and jobs within a policy region. However, if you do not have a script, you can avoid losing your tasks and jobs by performing the following procedure:

1. Delete the tasks to be replaced in the task library from the desktop or by using the wdeltask command as follows:
   
   ```
   wdeltask MyTask TaskLib
   ```
   
   where `MyTask` is the name of the task and `TaskLib` is its task library.

2. Create a copy of the tll file named tll2, using the following command:

   ```
   cp tll tll2
   ```

   Edit the new file (tll2).

3. Delete everything in the new file except the ArgLayout clauses and task definitions. For additional information on task definitions and ArgLayout clauses, see "Defining Argument layouts" on page 441 and "Creating task definitions" on page 451, respectively.

4. Run the following command to load the new tasks into the task library.

   ```
   wtll -i -l TaskLib -P /usr/ccs/lib/cpp tll2
   ```

   For more information, see "wtll" on page 378.

### Using message catalogs with a task library

The TLL uses many of the same features available within the Dialog Specification Language (DSL). DSL is part of the Tivoli Application Developer’s Environment (ADE) that provides the services to create a GUI.

From a TLL perspective, message catalogs are one of the most important features of DSL. Message catalogs enable you to display the text found on a window in multiple languages. *TME 10 ADE Desktop Services Manual* and *TME 10 ADE Application Services Manual, Volume I* describe how to create and implement a message catalog.

Message catalog references in TLL definition use the following format:

`'MsgCatalogName', 'DefaultText', 'Key'

where:

*MsgCatalogName*
  Specifies the message catalog to contain the message to be displayed.
If no message catalog is used, then use " " as the message catalog name. Any string enclosed within quotation marks, such as " _!_ " , can be used to indicate that there is no message catalog. However, the task library processes the value an empty string faster.

**DefaultText**

Specifies the text to display when the message catalog cannot be found or does not exist.

**Key**

Specifies a number that identifies an entry in the message catalog.

**Note:** The number 1 is used in this book to indicate that a message catalog is not used.

In addition, some TLL elements use a return value that can be the same as the **DefaultText**. In the following example, the first instance of the words **Ascending** and **Descending** are the values that are displayed on the window. The second instances are values that are returned to the script that performs the task:

```c
RadioButton {
    { ("" , "Ascending",1) "Ascending" }
    { ("" , "Descending",1) "Descending" }
}
```

To improve the readability of your task definition, use return values that are different than the displayed text. The following example illustrates this principle:

```c
RadioButton {
    { ("" , "Ascending",1) "ASC" }
    { ("" , "Descending",1) "DESC" }
}
```

## Enabling tasks to be run on endpoints

Most tasks require a shell, the Perl interpreter, or similar tool. These tools are usually available on managed nodes, and tasks should be able to run. On PC endpoints, however, these tools are frequently unavailable.

The `$BINDIR/tools` directory is created during the installation of a managed node. This directory contains a Perl interpreter (**perl.exe**) as well as command and utilities ported from UNIX, including the Bourne shell (**sh.exe**), the **awk** command (**awk.exe**), and the **grep** command (**grep.exe**).

When an endpoint is created on Windows operating systems, the tools directory, `$BINDIR/tools`, is not included. If you have tasks that need access to these tools, you must create a dependency to download the tool on demand.

This section describes how to establish this dependency. Although the procedure describes the process for Windows-based endpoints, you might need to perform this procedure for other operating systems.

**Notes:**

- You can create dependencies from any gateway, but it is easier to set up a dependency using an Windows gateway (because the Windows-specific tools are installed only on Windows managed nodes). UNIX managed nodes, have no copy of the Windows-specific tools installed. To install these tools on a UNIX managed node, an additional step is required to set up the tool dependencies.
• There is no way for applications to specify dependencies on a task-by-task basis. For a task library, the solution is to put the dependency on the `run_task` method.

Perform the following steps to set up dependencies for task Library scripts on Windows endpoints:

1. For a Windows gateway, copy the from the $BINDIR/tools directory to the Windows gateway.

2. Run the `wdepset` command with the `-c` option to create the dependency set. The most basic implementation of this command is as follows:

   ```
   wdepset -c label -a interp path +p path
   ```

   where:

   `-c label`  
   Creates a dependency set with the specified label.

   `-a interp`  
   Specifies the interpreter type of the endpoint.

   `path`  
   Specifies the path to the tool. The path is relative to the interpreter type specified in the `-a` option.

   `+p path`  
   Indicates the dependent file is not to be deleted from the endpoint when the endpoint cache becomes full. The `path` option indicates the directory on the endpoint to use as the base directory for downloading the dependency. For the Windows tools, the `path` option can be specified as the `%TOOLS%` environment variable.

   The following is an example of the `wdepset` command:

   ```
   wdepset -c task-tools
   -a win95 bin/win95/tools/sh.exe +p %TOOLS% \
   -a w32-ix86 bin/w32-ix86/tools/sh.exe +p %TOOLS% \
   -a w32-ix86 bin/w32-ix86/tools/win32gnu.dll +p %TOOLS%
   ```

3. Use the `wchdep` command to associate the dependency with the `run_task` method.

   ```
   wchdep @Classes:TaskEndpoint @DependencyMgr:task-tools run_task
   ```

   For task libraries, the only variable in the `wchdep` command is the name of the dependency. In this case, `task-tools` is the dependency.

4. Use the `wgateway` command with the `dbcheck` option to synchronize the gateway cache with the Tivoli server. For example, if your gateway is named subnet30-gateway, enter the following command:

   ```
   wgateway subnet30-gateway dbcheck
   ```

For more information about dependencies, see *Tivoli Application Development for the Lightweight Client Framework*. 
Chapter 9. Syntax for the task library language

When you create a task from the Tivoli desktop, you do not have access to all features provided by the task library language (TLL). The most important feature that the TLL provides is the ability to construct windows that enable users to provide information. To enable these features, you must be familiar with the syntax of a task library definition.

A task library definition contains the following sections:

**Library characteristics**
This section defines several global attributes, including the name of the task library and its version number. This section always appears at the beginning of the task library definition.

**Argument layouts**
This section defines each Argument layout. An Argument layout defines an element on a window, such as a text box, a scrolling list, and so forth. The values that users provide or select are passed to the appropriate task definition. An Argument layout can be placed before or after a task definition.

**Task definitions**
This section defines the task definition, which includes the code that implements the task. It also uses the Argument layouts to construct windows, enabling a user to provide information that is passed to the task definition as arguments. A task definition is much more complex than specifying library characteristics or defining Argument layouts.

### Setting library characteristics

The library characteristics section must be the beginning of the task library definition. It defines characteristics that are global to the entire task library. These characteristics include the task library name, the help message that is displayed for the library, and the version number.

The characteristics section of a task library begins with the following statement:

```
TaskLibrary library_name {
```

where `library_name` is the label for the task library.

In addition, the library characteristics section contains some or all of the following attributes. These attributes provide information about how the library interacts with other parts of the Tivoli environment.

When you create a task library from the Tivoli desktop, Tivoli Management Framework provides the following code as a foundation:

```
#ifndef TASK_BINDIR
#define TASK_BINDIR "./"
#endif

TaskLibrary "NewTaskLibrary" {
    Context = ("-l ";"-u",1);
    Distribute = ("-l ","ALI",1);
```
HelpMessage = ("_!_","Conventional Task Library",1);
Requires = ("_!_",">2.5",1);
Version = ("_!_","1.0",1);
}

Note: The first three lines provide information to the C preprocessor about the location of the task library directory. It is not necessary to add these lines when you create a new task library.

The TaskLibrary statement begins the general description of the task library and must be present in all task libraries. The "NewTaskLibrary" argument is the label of the task library. If you change this label, a new task library is created when you import the edited task library back into the Tivoli environment.

The values listed in the default code sample use the format ("_!_","Text","1"). This format corresponds to the message catalog format defined in the dialog specification language (DSL). If you do not plan to translate or localize the task library, the only important value is the string specified by the Text option.

The following keywords help define the characteristics of a task library. They can be used in any order before the Argument layout and task definition sections of the task library.

Context
Specifies the application that uses the task library. Applications use this attribute to filter the list of installed task libraries. This attribute is optional.

Distribution
Indicates where the task library binaries are stored. The value for this keyword is relevant only if distributed monitoring tasks defined in the task library.

The following values are valid:

ALI Specifies that the task binaries are stored only on the Tivoli server for the local Tivoli region.

GLOBAL Distributes copies of the task binaries to every managed node in every connected Tivoli region.

LOCAL Distributes copies of the task libraries to every file server in the local Tivoli region. If the task library is distributed to multiple machines, and if the path is a shared file system, the task library is distributed one time.

HelpMessage
Defines a help message string. This attribute is optional.

Requires
Specifies the version of Tivoli Management Framework required to run the task library. If a greater than sign (>) is specified before the version number, then any software version greater than the stated version is also supported. This attribute is optional.

Version
Defines a string that describes the version of the task library. This attribute is required.
Defining Argument layouts

When you create a task using only the features available from the Tivoli desktop, you cannot customize the windows associated with the tasks. As a result, the tasks you write must be self-contained, requiring no user input. However, the TLL allows you to define Argument layouts to create a window that contains text boxes, radio buttons, and so forth. The values the users provides in the window can be passed to the task as arguments.

The basic syntax of an Argument layout is as follows:

```
ArgLayout Name {
  LayoutType;
  ButtonLabel ("Text");
};
```

The `Name` variable must be specified for each Argument layout. This name is then referenced in Argument clauses that are contained in task definitions. For more information about Argument clauses, refer to "Defining Argument clauses" on page 454.

The `LayoutType` argument is a TLL keyword that must have one of the following values. Each of these layouts provides the basic structure of a gadget.

- Text
- CryptoText
- Choice
- TextChoice
- ChoiceButton
- RadioButton

The `ButtonLabel` keyword defines the text displayed on a button. You must include the `ButtonLabel` keyword with all of the layout types that use buttons.

The window that contains the gadgets is run as a non-privileged user. If the window contains scripts that need to be run as either the privileged user, or as the calling administrator, the scripts must be installed as custom methods. For more information, see "Defining an Implementation clause" on page 452.

The following sections describe each of the layouts. Sample code and a snapshot of the window the code produces are included.

**Note:** The Argument clauses listed in each example are not part of the layout definitions. They are included in the task definitions. They are listed here to show how they are used with layouts to form a window. Refer to "Defining Argument clauses" on page 454 for explanations of the keywords used within an Argument clause.

### Text and CryptoText

The **Text** layout creates a text box that users can type directly into. The **CryptoText** layout is similar to the **Text** layout, except the text a user enters is masked with characters such as asterisks (*). The **CryptoText** layout can be used for password fields or other fields where standard text should not be displayed.

**Note:** The text entered in a **CryptoText** layout is not encrypted. The text is transmitted as clear text.
A **Text** and **CryptoText** layouts are defined as follows:

```plaintext
ArgLayout TextField {
    Text;
};

ArgLayout PasswordField {
    CryptoText;
};
```

The **TextField** and **PasswordField** arguments are identifiers referenced in the Argument clause of a task. The following code samples illustrate how layouts and arguments are related:

```plaintext
Argument (**", "User Name", 1) {
    Layout = (**", "TextField", 1);
};

Argument (**", "Password", 1) {
    Layout = (**", "PasswordField", 1);
};
```

If these Argument statements are placed in the same task definition, a window similar to the following is displayed:

![Window](image)

Text is displayed in the text fields as follows:

![Window](image)

The **Text** and **CryptoText** layouts do not use any additional attributes. Because there are no customized attributes for these layouts, you can define a single instance of these layouts and reuse them many times within a task library.
Choice and TextChoice

The Choice layout creates a button that launches a choice list window. When a user selects an item from the choice list, the item is displayed in a non-editable field of the task window.

The TextChoice layout adds an editable text box to the features the Choice layout provides. The user can either use the choice list window or enter text into the text box. If the user later selects an item from the choice list window, the contents of the text box are overwritten.

The following code samples illustrate the Choice and TextChoice layouts:

```csharp
ArgLayout SelectManNode {
    Choice Resource "ManagedNode"
    ButtonLabel = ("", "ManagedNodes...", 1);
};
ArgLayout SelectFrequency {
    TextChoice Multi ("", ",", ",") File "/bin/scripts/Times.txt"
    ButtonLabel = ("", "Frequency...", 1);
};
```

The Multi keyword enables the user to choose multiple items from the select list. (The user must press the Ctrl key to select multiple items.) This keyword must be placed directly after the Choice or TextChoice keyword, separated by a space. The Multi keyword must also be followed by a set of instructions that indicate how the selected items will be delimited. These instructions must be in the following format:

("leading_string", "separator_string", "trailing_string")

Example delimiters include commas and colons, but you can use any character your script can process. In the ArgLayout SelectFrequency example (above) there are no leading and trailing strings, but the separator string is set to a comma followed by a space. In the following example, the first selected item in a list is preceded with a left parenthesis, each item is separated by a colon, and the last item is followed by a right parenthesis.

```csharp
ArgLayout ManagedNode {
    Choice Multi ("(", ",:", ")") File "/bin/scripts/Times.txt"
    ButtonLabel = ("", "Frequency...", 1);
};
```

The items in a choice list can be derived dynamically, read from a file, or explicitly coded into the layout definition. The Resource keyword is one method of populating a choice list dynamically. It generates a list of instances of the specified Tivoli resource defined within the current Tivoli region, such as ManagedNode or Administrator.

The TLL accepts the following keywords for generating choice lists:

**File** "Path"

Opens the specified file and displays each line in the file as a selectable option in the choice list. The file cannot have comments or other extraneous information. It must be placed on the TMR server or other location so that the task engine can find it.

**FileBrowser**

Displays a file browser window instead of a choice list. The browser will display files on the Tivoli server. The DefaultValue keyword specifies the
path and host initially displayed. See "Defining Argument clauses" on page 454 for more information about the DefaultValue keyword.

**FileData "Path"**

Opens the specified file and displays each odd-numbered line in the file as a selectable option on the choice list. The even-numbered lines are return values. When an item is selected, the return value is displayed on the task window.

**HostFileBrowser**

Displays a file browser that also enables you to select the host the file is stored on. The DefaultValue keyword specifies the path and host initially displayed. See "Defining Argument clauses" on page 454 for more information about the DefaultValue keyword.

**Program Script**

Runs a program or script and uses each line of output as the choices. The programs and scripts must follow the same syntax and conventions as those defined in the task definitions. See "Defining an Implementation clause" on page 452 for more information.

**ProgramData Script**

Follows the same conventions as the Program keyword, except odd-numbered lines are displayed as selectable options, and even-numbered lines are return value. When an item is selected, the return value is displayed on the task window.

**Resource "Name"**

Queries the Tivoli name registry and returns a list of instances that match the specified resource name. Each instance is displayed as a selectable option. The resource name can be any item that is displayed by running the wlookup command with the –R option.

Note: To view a list of all administrators in an environment with connected Tivoli regions, specify the AdministratorCollection resource rather than Administrator.

**ResourceOID "Name"**

Queries the Tivoli name registry and returns a list of instances that match the specified resource object ID. Each instance is displayed as a selectable option.

You can also define a set of selectable options within the layout. This method of generating options does not use a keyword, but must use the following syntax:

```c
{ ("Msgcat","DisplayedValue",Key) "ReturnValue"
  { ... }
};
```

The Msgcat, DisplayedValue, and Key arguments are required even if you are not translating your task library. The ReturnValue argument is the value that is returned to the task for processing.

**Example—Choice layout with the Resource keyword**

The SelectManNode layout creates a choice list with a button that has the ManagedNodes... label. The text to the left of the button is defined in the Argument clause.
ArgLayout SelectManNode {
    Choice Resource "ManagedNode";
    ButtonLabel = ("", "ManagedNodes...", 1);
};

Argument ("", "Select a Managed Node", 1) {
    Layout = "SelectManNode";
};

The following window is displayed when the user selects the task containing this Argument clause:

![Window](image1.png)

When you click ManagedNodes, a window similar to the following is displayed:

![Window](image2.png)

Because the layout definition did not include the Multi keyword, the label above the select list is Choose one. When you select an option and dismiss the window, the selected item is displayed in a non-editable text box to the right of the button.
Example—TextChoice layout with the File keyword

The following example uses the File keyword to generate a list of options from a text file. The task engine reads each line in the specified text file and places it in the option list.

ArgLayout SelectFrequency {
    TextChoice Multi ("", ",", "") File "/bin/scripts/Times.txt"
    ButtonLabel = ("", "Frequency...", 1);
};

Argument ("", "Frequency", 1) {
    Layout = "SelectFrequency";
};

The contents of the Times.txt file could be similar to the following:

1
5
10
25

The following window is displayed when the user selects the task containing this Argument clause:

![Window Displaying Options]

When the user clicks Frequency, a window similar to the following is displayed:

![Window with Options]

Because the Multi keyword was used in the layout, the label for the option list is Choose one or more. When a user selects one or more items and dismisses the window, the task window displays the selections in a text box to the right of the button.
Example—TextChoice layout with an embedded Choice List
To specify the options defined in the Times.txt file without referring to the file, the layout definition would be similar to the following:

```plaintext
ArgLayout ListOfNumbers {
    Choice ("", "", "", "")
    {{("", "1", 1) "1"}
    {{("", "5", 1) "5" }
    {{("", "10", 1) "10" }
    {{("", "25", 1) "25" }};
    ButtonLabel = ("", "Frequency...", 1);
};
Argument ("", "Frequency", 1) {
    Layout = "ListOfNumbers";
};
```

**ChoiceButton**

The **ChoiceButton** layout creates a pop-up menu. Only one item can be selected from a pop-up list. The TLL accepts the following keywords for generating a pop-up list:

- **File “Path”**
  Opens the specified file and displays each line in the file as a menu option. The file cannot have comments or other extraneous information. It must be placed on the Tivoli server or other location so that the task engine can find it.

- **FileData “Path”**
  Opens the specified file and displays each odd-numbered line in the file as a menu option. The even-numbered lines are return values.

- **Program Script**
  Runs a program or script and uses each line of output as menu options. The programs and scripts must follow the same syntax and conventions as those defined in the task definitions. See “Defining an Implementation clause” on page 452 for more information.

- **ProgramData Script**
  Follows the same conventions as the **Program** keyword, except odd-numbered lines are displayed as menu options, and even-numbered lines are return values.

- **Resource “Name”**
  Queries the Tivoli name registry and returns a list of instances that match the specified resource name. Each instance is displayed as a menu option.
Example—Program keyword
The Program keyword allows you to run a script to create a list of menu options. In this example, a script reads from the /etc/hosts file to generate a list of hosts.

Note: The Implementation keyword is used in the following examples. For more information about this keyword, see “Defining an Implementation clause” on page 452.

ArgLayout Hosts {  
    ChoiceButton Program {  
        Implementation ("default")  
        .#!/bin/sh  
        .awk < /etc/hosts '/^\[0-9][0-9]*/ {print $2}' | sort  
        .exit 0  
    };  
};

Argument ("", "Select Host", 1) {  
    Layout = ("", "Hosts", 1);  
};

The following window is displayed:

Example—Resource keyword
The following example creates a pop-up menu containing the names of the Tivoli administrators.

ArgLayout Admin {  
    ChoiceButton Resource "Administrator";  
};

Argument ("", "Select Administrator", 1) {  
    Layout = ("", "Admin", 1);  
};

The following window is displayed:
RadioButton

The RadioButton layout creates a set of radio buttons. These differ from choice buttons in that only one radio button can be selected at a time. Like choice buttons, radio buttons can be derived dynamically, read from a file, or explicitly coded into the layout definition.

The TLL accepts the following keywords for generating a set of radio buttons:

**File “Path”**
Opens the specified file and displays each line in the file as a radio button. The file cannot have comments or other extraneous information. It must be placed on the Tivoli server or other location so that the task engine can find it.

**FileData “Path”**
Opens the specified file and displays each odd-numbered line as a radio button. The even-numbered lines are return values.

**Program Script**
Runs a program or script and uses each line of output as a radio button. The programs and scripts must follow the same syntax and conventions as those defined in the task definitions. See “Defining an Implementation clause” on page 452 for more information.

**ProgramData Script**
Follows the same conventions as the Program keyword, except odd-numbered lines are displayed as radio buttons, and even-numbered lines are return values.

**Resource “Name”**
Queries the Tivoli name registry and returns a list of instances that match the specified resource name. Each instance of the resource is displayed as a radio button.

You can also define a set of radio buttons within the layout. This method of generating options does not use a keyword, but must use the following syntax:

```plaintext
{ 
    {("Msgcat","DisplayedValue",Key) "ReturnValue"}
    { ... } 
};
```
The `Msgcat`, `DisplayedValue`, and `Key` arguments are required even if you are not translating your task library. The `ReturnValue` argument is the value that is returned to the task for processing.

**Example—File keyword**

This example references the same file used in the example of a the `TextChoice` layout. However, each line in the source file is displayed as a radio button, rather than an item in an option list.

```
ArgLayout FrequencyButtons {
    RadioButton File "/bin/scripts/Times.txt";
};
Argument ("", "How many times to run this script?", 1) {
    Layout = ("", "FrequencyButtons", 1);
};
```

The following window is displayed:

![Example window](image)

**Example—embedded list of radio buttons**

The `SeverityButtons` layout creates a set of radio buttons that correspond to the severities of events. The first item is selected by default.

```
ArgLayout SeverityButtons {
    RadioButton {
        { ("", "High", 1) "HIGH" }
        { ("", "Medium", 1) "MEDIUM" }
        { ("", "Low", 1) "LOW" }
    };
};
Argument ("", "Select Severity", 1) {
    Layout = ("", "SeverityButtons", 1);
};
```

The following window is displayed:
Creating task definitions

A task definition contains either two or three sections. The first section is the task header, which specifies basic characteristics of the task, such as the authorization role and user ID needed to execute the task. The second section provides a script or pointer to a file that implements the task. The third section defines the arguments. The argument definitions and the Argument layouts construct the window that is displayed.

Defining a task header

The task header section always begins with the Task keyword followed by the task name and left brace ({}). The task name must be unique within the task collection and cannot contain spaces, quotation marks, or be a TLL keyword.

Next, a set of attribute and value pairs define characteristics of the task. The following attributes are valid:

Comments
Provides commentary about the task that is not displayed on a window. The comments must be enclosed in double quotation marks (" "). This attribute is optional.

Description
Specifies a brief description of the task, which is displayed on the Configure Task Arguments window.

Gid
Specifies the name of the group (group ID) under which the task will run. This attribute is optional.

HelpMessage
Specifies the help message that is displayed when the user clicks Task Description. This attribute is optional.

The help message is a simple text string. Each line within the help message should be limited to 80 characters or fewer. You can use the following control characters to determine how text in the help message is displayed:
\n Forces subsequent text to be displayed on a new line.
\r Inserts a blank line.
\t Inserts a horizontal tab.
Roles  Specifies the authorization roles required to run the task. Multiple roles must be separated by colons, such as Roles = "admin:senior:super". This attribute is required.

Uid    Specifies the name of the user under which the task will run. Use an asterisk enclosed in quotation marks ("*") to set the user ID to the current user. This attribute is optional.

The following example illustrates the header information for a task:

```plaintext
Task CheckFileSystem {
    Description = ("", "Checks the availability of a file system", 1);
    HelpMessage = ("", "This task runs the ls command to determine to determine whether the file system is available. You must specify a directory name and user name to execute this task.", 1);
    Uid = ("", "", 1);
    Roles = ("", "senior:admin:user", 1);
    Comments = ("", "Created 1/17/98", 1)
}
```

**Defining an Implementation clause**

The Implementation clause defines the script that performs the work of the task. The Implementation clause also specifies the interpreter types the script is designed for. A task can have several Implementation clauses.

The script can be a Bourne shell script, a Perl script, a Tivoli command, or similar executable. The script must be able to run on the desired target. Windows endpoints, for example, do not automatically have access to the Bourne shell or a Perl interpreter. A dependency set must established for these endpoints. See "Enabling tasks to be run on endpoints" on page 437 for more information.

**Note:** For a list of supported interpreter types, see Tivoli Management Framework Release Notes. The interpreter types correspond to the list displayed on the Create Task window on the Tivoli desktop.

The script can be embedded in the Implementation clause, or the clause can point to an external file or program. An Implementation clause can be defined as a main clause within a task definition, or as a subclause within the Program and ProgramData Argument layouts.

The syntax of the Implementation clause depends on whether the script is external or embedded. For external scripts, the syntax is as follows:

```
Implementation ("Interp") Binary "File";
```

The Binary keyword indicates that the task code is located in a separate file, and the File argument specifies the path to that file. When you import the task library back into the Tivoli environment, the task engine copies the script file into the directory specified by the TASK_BINDIR variable. (The TASK_BINDIR variable is assigned at the beginning of the task library definition.) The engine also renames the file to number.default, such as 0.default. Any subsequent changes to the script should be made in the renamed file.

The following example illustrates the use of the Binary keyword:

```
Implementation ("aix4-rl") Binary "aix_list.sh";
Implementation ("default") Binary "list.sh";
```
**Note:** The TLL was designed to implement embedded shell scripts using `/bin/sh`. Other scripting languages, such as Perl, may not provide the expected results. As a result, use the **Binary** keyword to implement other types of scripts.

The syntax for embedding a script into the task is as follows:

```plaintext
Implementation ("Interp")
 .Line of Code
 .Line of Code
 ...
```

In a **Program** layout, after you have defined the name and type of the layout, the syntax for embedding a script is the same. The following sample defines a layout named **Hosts** and uses a **ChoiceButton**:

```plaintext
ArgLayout Hosts {
 ChoiceButton Program
   Implementation ("Interp")
   .Line of Code
   .Line of Code
 ...
};
```

Each line in an embedded script begins with a period (.). The TLL translator treats these lines as if they were a double-quoted string. Additional quotation marks are not required. This syntax makes it easier to include straight shell script code and reduces the potential problems involved with using quotes within shell scripts.

**Note:** Tasks that are imported into a task library with the **Binary** keyword do not need to be rewritten so that each line begins with a period.

Unlike the task itself, scripts and commands that are embedded into an Argument layout will run as a non-privileged user. If there is a need to have them run as either the privileged user, or as the calling administrator, they must be installed as custom methods.

To install a script as a custom method, it first needs to be copied under the directory pointed to by the `$BINDIR` variable. It is recommended to first create a subdirectory for custom scripts, as illustrated in the following example:

```plaintext
mkdir $BINDIR/CUSTOM
 cp /path/to/mymethod.sh $BINDIR/CUSTOM/mymethod.sh
```

The **wputmeth** command installs the custom method. To install the script to run as root, use the following command:

```plaintext
wputmeth -u root -g other -r TaskLibrary mymethod /CUSTOM/mymethod.sh
```

To install the script to run as the calling administrator, use the following command:

```plaintext
wputmeth -u '*' -g '*' -r TaskLibrary mymethod /CUSTOM/mymethod.sh
```

In the **Implementation** section of the Argument layout, the method would be called using the **objcall** command.

```plaintext
Implementation ("Interp")
 .objcall 'wlookup -r TaskLibrary LibName' mymethod
```

where **Interp** is a valid interpreter type and **LibName** is the name of the library being installed.
Defining Argument clauses

The Argument clause uses an Argument layout (ArgLayout) to create a gadget on the task window. This clause reads the data associated with an Argument layout and passes this information to the task script. Therefore, you must define an Argument clause for each argument defined in your script. In addition, the Argument clauses must be defined in the order that the gadgets are displayed in the window (top to bottom).

The following syntax describes the Argument clause:

```
Argument "ArgumentName" {  
    Layout = "ArgLayoutName";
    OptionalAttributes;
}
```

The value specified by the ArgumentName parameter is a label that instructs the user what type of information to enter or select. The label is displayed to the left of the gadget. For example, if you are creating a text box that the user is required to type the path to a file, the value of the ArgumentName parameter should be something similar to File Name.

The Layout keyword is required, and its value must match a defined Argument layout. The task engine cannot display a gadget unless the specified layout has been defined. The Argument clause also accepts the following attributes. All of these attributes are optional.

**ButtonLabel**
- Specifies the text to be displayed on a button.

**DefaultValue**
- Specifies a default value for the argument.
  - If the layout is Text or CryptoText, the default value is displayed in the text box.
  - If the layout is a Choice, TextChoice, ChoiceButton, or RadioButton, the specified value must match the return value of an option. The return value and displayed value of an option item can be different.
  - If the layout is a FileBrowser or HostFileBrowser, the browser window displays the specified directory.
  - If this attribute is not specified for a RadioButton or ChoiceButton layout, the first option is selected.

**MustMatch**
- Specifies a regular expression that any configured argument value must match.

**MustNotMatch**
- Specifies a regular expression the argument must not match.

**SectionLabel**
- Provides a group label that will be displayed above the argument containing the current attribute. You can include up to four SectionLabel attributes. If you need to provide additional information, you can create a help message using the HelpMessage attribute.

The following examples show how the MustMatch and MustNotMatch keywords can be used.
Checks at the beginning of a line to see if the string contains at least five alphanumeric characters, or checks if any string contains at least five alphanumeric characters and ends with -XYZ. The check for the XYZ substring is case insensitive.

Checks at the beginning of a line for an IP address starting with 120.22. The remaining two groups of numbers must contain one, two, or three digits.

Checks an alphanumeric string to determine if it has between three and eight characters.

Checks for dates in the format MM-DD-YYYY. Additional checking is required, because it is possible to enter invalid dates.

Checks for a correctly formatted Social Security Number, in the form NNN-NN-NNNN.

The following example illustrates several layouts and Argument clauses:

```plaintext
ArgLayout AscDesc {
    RadioButton { {{"", "Ascending", 1) "ASC"},
        {{"", "Descending", 2) "DESC"} ;
    
    ArgLayout Number {
        Text
    };

    Argument ("", "Ascending or Descending order?", 1){
        Layout = "AscDesc";
        DefaultValue = "DESC;"
    };

    Argument ("", "Number of Events", 1) {
        Layout = "Number";
        MustMatch = "^[0-9][0-9]*$";
        DefaultValue = ("","1",1);
    };
}
```

The default values for the arguments are DESC (Descending) and 1, respectively. The MustMatch keyword for the Number layout ensures that the value entered in the text field is an integer.

You can include a maximum of 23 arguments per task. If you require more than 23 arguments, create a dialog box using the Dialog Specification Language (DSL) file that contains the argument information that you require. You must compile and install the DSL dialog box before using it. For more information about DSL, see TME 10 ADE Desktop Services Manual and TME 10 ADE Application Services Manual, Volume I.

You can then create a task that invokes the DSL dialog box. To create a task that invokes a DSL dialog box, ensure that the Layout keyword in the Argument clause is set to Dialog, as shown in the following example:

```plaintext
Argument ("", "DisplayedValue", key
    Layout="Dialog;"

Chapter 9. Syntax for the task library language  455
Also, ensure that the name of the DSL dialog box has the same name as the task definition. For example, if your task is defined as follows, the name of the compiled and installed DSL dialog box must be RunSQL.d.

Task "RunSQL" {

When the Layout keyword in the Argument clause is set to Dialog, the task library code locates and runs a file with the same name as the task definition.

Adding comments to a task library

The wtll command does not recognize the C++ comment indicators (//) as valid TLL syntax. However, it does recognize the C comment indicators (/* */).

The C preprocessor can be configured to ignore all text on a line that follows the // characters. The following command allows you two use C++ comments in a task library definition:

wtll -i -l LibraryName -P /usr/ccs/lib/cpp FileName -B -P

The -B and -P options are part of the cpp command. The -B option enables the comment indicators, and the -P option provides additional processing instructions.

If you want to use C comment indicators, it is not necessary to specify the -B and -P options.

Editing the TLL configuration files

The task library engine uses a set of files to determine which arguments within a task library definition are optional. On UNIX operating systems, these files are located in the /etc/Tivoli/tll.conf directory. On Windows operating systems, they are in the %SYSTEMROOT%\system32\drivers\etc\Tivoli\tll.conf directory.

The tll.conf directory contains the following files:

<table>
<thead>
<tr>
<th>arg</th>
<th>Controls the parsing of the Arguments section. For more information about the attributes this file controls, see “Defining Argument clauses” on page 454.</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout</td>
<td>Controls the parsing of attributes within the ArgLayout section of a task definition. Currently, ButtonLabel is the only attribute supported.</td>
</tr>
<tr>
<td>library</td>
<td>Controls the parsing of the task library header. For more information about the attributes this file controls, see “Setting library characteristics” on page 439.</td>
</tr>
<tr>
<td>task</td>
<td>Controls the parsing of task headers. For more information about the attributes this file controls, see “Defining a task header” on page 451.</td>
</tr>
</tbody>
</table>

Attributes defined within these files must use the following conventions:

- Each attribute must be listed on a separate line. There are no semicolons or other punctuation marks at the end of the line.
- If the attribute is required, do not place a special character before the attribute name.
- A question mark (?) precedes the name of an optional attribute.
- An exclamation point (!) precedes the name of an attribute that is not allowed.
• A number sign (#) precedes the name of an attribute that will be ignored when the task library is parsed.

New attributes cannot be added to the tll.conf files.

The following example illustrates the default library file:

```
Version
?Requires
HelpMessage
?Context
Distribute
?DisplayCfgd
```

**Note:** The **DisplayCfgd** keyword is one of several keywords that are defined in the tll.conf files but are not implemented in task libraries.
Chapter 10. Sample Task Library

This chapter illustrates a sample task library. It contains most of the possible elements within the task library language.

TaskLibrary "T/EC Tasks" {
  Distribute = "ALI";
  Version = "1.0";
  Requires = ">2.1";
  Context = "TEC";

  HelpMessage = (tec_tasks, "Tasks for use by the IBM Tivoli Enterprise Console", 1);

  ArgLayout Filename {
    TextChoice FileBrowser;
    ButtonLabel = (tec_tasks, "File...", 2);
  };

  ArgLayout HostFilename {
    TextChoice HostFileBrowser;
    ButtonLabel = (tec_tasks, "File...", 2);
  };

  ArgLayout ManagedNode {
    Choice Multi ("", ",", ",", ) Resource "ManagedNode";
    ButtonLabel = (tec_tasks, "Managed Node...", 3);
  };

  ArgLayout Administrator {
    TextChoice Resource "AdministratorCollection";
    ButtonLabel = (tec_tasks, "Administrator ...", 131);
  };

  ArgLayout EventServer {
    TextChoice Resource "EventServer";
    ButtonLabel = (tec_tasks, "Choose ...", 68);
  };

  ArgLayout PolicyRegion {
    TextChoice Resource "PolicyRegion";
    ButtonLabel = (tec_tasks, "Choose ...", 68);
  };

  ArgLayout Host {
    TextChoice Program {
      Implementation ( "hpux9" "sunos4", "solaris2", "aix3-r2" )
      .#!/bin/sh
      . awk < /etc/hosts /^[0-9][0-9]*\{print $2\}'
    };
    ButtonLabel = (tec_tasks, "Hosts ...", 66);
  };

  ArgLayout FileText {
    RadioButton { ((tec_tasks, "File", 4) "FILE")
      ((tec_tasks, "Text", 5) "TEXT") };
  };

  ArgLayout AscDesc {
    RadioButton { ((tec_tasks, "ASC", 6) "ASC")
      ((tec_tasks, "DESC", 7) "DESC") };
  };

  ArgLayout YesNo {
    RadioButton { ((tec_tasks, "Yes", 8) "YES")
      ((tec_tasks, "No", 9) "NO") };
  };
}
ArgLayout Number {
  Text
};

ArgLayout SimpleText {
  Text;
};

ArgLayout Severities {
  ChoiceButton {
    
    
    
    
    
  }
};

ArgLayout SelectSeverity {
  TextChoice Multi ("", ",", ",")
  {{(tec_tasks, "UNKNOWN", 69) "UNKNOWN"}
   {{(tec_tasks, "HARMLESS", 70) "HARMLESS"}
   {{(tec_tasks, "WARNING", 71) "WARNING"}
   {{(tec_tasks, "MINOR", 72) "MINOR"}
   {{(tec_tasks, "CRITICAL", 73) "CRITICAL"}
   {{(tec_tasks, "FATAL", 74) "FATAL"}};
  ButtonLabel = (tec_tasks, "Severity ...", 75);
};

ArgLayout SelectStatus {
  TextChoice Multi ("", ",", ",")
  {{(tec_tasks, "OPEN", 76) "OPEN"}
   {{(tec_tasks, "RESPONDED", 77) "RESPONDED"}
   {{(tec_tasks, "ACKNOWLEDGED", 78) "ACKNOWLEDGED"}
   {{(tec_tasks, "CLOSED", 79) "CLOSED"}};
  ButtonLabel = (tec_tasks, "Status ...", 80);
};

ArgLayout SelectSlots {
  Choice Multi ("", ",", ",")
  {{(tec_tasks, "server_handle", 81) "server_handle"}
   {{(tec_tasks, "date_reception", 82) "date_reception"}
   {{(tec_tasks, "event_handle", 83) "event_handle"}
   {{(tec_tasks, "source", 84) "source"}
   {{(tec_tasks, "sub_source", 39) "sub_source"}
   {{(tec_tasks, "origin", 36) "origin"}
   {{(tec_tasks, "sub_origin", 38) "sub_origin"}
   {{(tec_tasks, "hostname", 35) "hostname"}
   {{(tec_tasks, "adapter_host", 89) "adapter_host"}
   {{(tec_tasks, "status", 90) "status"}
   {{(tec_tasks, "administrator", 91) "administrator"}
   {{(tec_tasks, "acl", 92) "acl"}
   {{(tec_tasks, "severity", 37) "severity"}
   {{(tec_tasks, "date", 94) "date"}
   {{(tec_tasks, "duration", 95) "duration"}
   {{(tec_tasks, "msg", 96) "msg"}
   {{(tec_tasks, "msg_catalog", 97) "msg_catalog"}
   {{(tec_tasks, "msg_index", 98) "msg_index"}
   {{(tec_tasks, "num_actions", 99) "num_actions"}
   {{(tec_tasks, "credibility", 100) "credibility"}
   {{(tec_tasks, "repeat_count", 101) "repeat_count"}
   {{(tec_tasks, "cause_date_reception", 102) "cause_date_reception"}
   {{(tec_tasks, "cause_event_handle", 103) "cause_event_handle"}};
  ButtonLabel = (tec_tasks, "Slots ...", 104);
};

ArgLayout SelectSource {
  TextChoice Multi ("", ",", ",")
  Program {
    Implementation ( "default")
  }
}
#!/bin/sh
.EVENT_SERVER=`wlookup -r EventServer -a awk '{print $2}'`
.SOURCES=`idlcall $EVENT_SERVER list_sources`
for SOURCE in `echo $SOURCES | sed s/\[{}\]//g`
do
  if [ "$PRINT" = "YES" ];then
    echo $SOURCE | sed s/\"//g
  else
    PRINT=YES
    fi
.done
.exit 0
;
ButtonLabel = (tec_tasks, "Source ...", 105);

ArgLayout SelectClass {
  TextChoice Multi ("", ",", ",")
  Program {
    Implementation ( "default" )
    .#!/bin/sh
    .EVENT_SERVER="wlookup -r EventServer -a \ awk '{print $2}''
    .RB="idlcall $EVENT_SERVER_get_current_rb"
    .CLASSES="idlcall $EVENT_SERVER_list_rb_classes $RB"
    .for CLASS in `echo $CLASSES | sed s/\[{}\]//g`
do
      if [ "$PRINT" = "YES" ];then
        echo $CLASS | sed s/\"//g
      else
        PRINT=YES
      fi
    .done
    .exit 0
  }
};
ButtonLabel = (tec_tasks, "Event Classes ...", 106);

ArgLayout SelectHost {
  TextChoice Program {
    Implementation ( "default" )
    .#!/bin/sh
    .awk < /etc/hosts '/^[0-9][0-9]*\ [0-9]\ [0-9]*\ [0-9]*\ / {print $2}' | sort
    ;
  }
};
ButtonLabel = (tec_tasks, "Hosts ...", 66);

ArgLayout Order {
  RadioButton {
    (tec_tasks, "Ascending", 107) "ASC"
    (tec_tasks, "Descending", 108) "DESC"
  }
};

Task Send_Email {
  Description = (tec_tasks, "e-mail event info ", 16);
  HelpMessage = (tec_tasks, "This task will send information about the event to the supplied address. Provide the name and the email address of the administrator that you want to send the message to.", 17);
  Roles = "super:senior:admin:user";
  Argument (tec_tasks, "Administrator's Name", 18) {
    Layout = "SimpleText"
    DefaultValue = "Chris Sanders"
  };
  Argument (tec_tasks, "Administrator's E-mail Address", 19) {
    Layout = "SimpleText"
  };
}
Implementation ("default")
.
.PATH=/bin:/usr/bin:/usr/ucb:/usr/lib
.export PATH
.
.MAIL_MESSAGE="
    cause_date : $cause_date
    cause_hndl : $cause_hndl
    class_name : $class_name
    date_event : $date_event
    ev_key : $ev_key
    formatted_date : $formatted_date
    handle : $handle
    srvr_handle : $srvr_handle"
.
.for SLOT in "$SLOTS"; do
    eval slotVal="\"$SLOT\""
    if [ ! -z "$slotVal" ]; then
        length=`expr "$SLOT" : '.*'`
        if [ $length -gt 7 ]; then
            if [ $length -gt 15 ]; then
                MAIL_MESSAGE="$MAIL_MESSAGE
                SLOT : $slotVal"
            else
                MAIL_MESSAGE="$MAIL_MESSAGE
                SLOT : $slotVal"
            fi
        else
            MAIL_MESSAGE="$MAIL_MESSAGE
            SLOT : $slotVal"
        fi
    fi
.done
.
.echo Sending message to $2...
.sendmail -F 'T/EC' -t <<< __EOF__
    To: $2
    Subject: Automatic Event Response message
    Dear $1,
    .
    Event received with the following:
    .
    $MAIL_MESSAGE
    .
    Sincerely,
    .Event Task Send_Email
    __EOF__
    .
.echo done.

Task Log_Event_To_File {

Description = (tec_tasks, "Log event to file", 20);

HelpMessage = (tec_tasks, "This task will log information about an event to a
text file. Supply the path to the file that you want the information logged to.", 21);

Roles = "super:senior:admin";
Argument (tec_tasks, "Log File Name", 22) {
  Layout = "Filename";
  ButtonLabel = (tec_tasks, "Log File Name", 22);
};

Implementation ("default")
./bin/sh

.LOG_MESSAGE="
  .cause_date : $cause_date
  .cause_hndl : $cause_hndl
  .class_name : $class_name
  .date_event : $date_event
  .ev_key : $ev_key
  .formatted_date : $formatted_date
  .handle : $handle
  .srvr_handle : $srvr_handle"

.for SLOT in $SLOTS; do
  eval slotVal=""${SLOT}""
  if [ ! -z "$slotVal" ]; then
    length=`expr "$SLOT" : '.'`
    if [ $length -gt 7 ]; then
      LOG_MESSAGE="$LOG_MESSAGE
        SLOT : $slotVal"
    else
      LOG_MESSAGE="$LOG_MESSAGE
        SLOT : $slotVal"
    fi
  else
    LOG_MESSAGE="$LOG_MESSAGE
        SLOT : $slotVal"
  fi
  fi
.done

 echoed Logging event to file $1 ...

 echoed 

# Event received at `/bin/date`

." >> $1

.sync
done
.exit 0
};

Task Popup_Message {
Description = (tec_tasks, "Popup message on desktop ", 23);
HelpMessage = (tec_tasks, "This task will display a dialog on a
Tivoli desktop. Supply the administrator label and the path to the file
that contains the text to be displayed.", 24);
Roles = "super:senior:admin"

Argument (tec_tasks, "Administrator Name", 18) {
  Layout = "Administrator";
  MustMatch = ".";
};
Argument (tec_tasks, "File Name", 25) {
  Layout = "HostFilename";
  ButtonLabel = {tec_tasks, "File Name", 25};
};

Implementation ("default")
  #!/bin/sh
  .echo Posting message to an administrator desktop ...
  .wsendresp $1 $2
  .rc=$?
  .echo done
  .exit $rc
};
Task Wake_Up_Netscape {
  Description = {tec_tasks, "Jump Netscape to URL", 26};
  HelpMessage = {tec_tasks, "This task causes a Netscape window to be displayed with the specified URL. The window is from an already running Netscape process or a new Netscape process is started. Specify the URL to be displayed.", 27};
  Argument (tec_tasks, "URL to DISPLAY", 28) {
    Layout = "SimpleText";
    DefaultValue = "http://www.tivoli.com";
  }
  Roles = "super:senior:admin";
  Implementation ("default")
    #!/bin/sh
    .# Uncomment the following lines to assist with debugging
    .# exec > /tmp/view_online_doc.out 2>&1
    .# set -x
    .
    .# env
    .# echo $1
    .echo Sending command to netscape ...
    .if [ "$WD_DISPLAY" != "" ];then
      .DISPLAY="$WD_DISPLAY"
    .fi
    .export DISPLAY
    .PATH=$PATH:/usr/local/bin
    .
    .HOSTNAME=`uname -n`
    .URL=$1
    .netscape -noraise -remote "openURL($URL,new-window)"
    .netrc=$?
    .echo done
    .exit $netrc
};
Task Forward_Event {
  Description = {tec_tasks, "Forward event to event server", 29};
  HelpMessage = {tec_tasks, "This task forwards an event to another event server. Select the server to receive the event.", 30};
  Roles = "super:senior:admin:user";
  Argument (tec_tasks, "Event server name", 33) {
    Layout = "EventServer";
    MustMatch = ".";
  };
}
 Implementation ("default")
  .#!/bin/sh
  .
  .COMMAND="wpodestmsg -S $1"
  .
  .for SLOT in $SLOTS; do
    .  if [ $SLOT = server_handle ]; then
      .    continue
    .  fi
    .  if [ $SLOT = date_reception ]; then
      .    continue
    .  fi
    .  if [ $SLOT = event_handle ]; then
      .    continue
    .  fi
    .  if [ $SLOT = acl ]; then
      .    continue
    .  fi
    .  if [ $SLOT = duration ]; then
      .    continue
    .  fi
    .  if [ $SLOT = num_actions ]; then
      .    continue
    .  fi
    .  if [ $SLOT = credibility ]; then
      .    continue
    .  fi
    .  if [ $SLOT = repeat_count ]; then
      .    continue
    .  fi
    .  if [ $SLOT = cause_date_reception ]; then
      .    continue
    .  fi
    .  if [ $SLOT = cause_event_handle ]; then
      .    continue
    .  fi
    .  if [ $SLOT = administrator ]; then
      .    continue
    .  fi
    .    eval slotVal="\"${SLOT}\""
    .    if [ ! -z "$slotVal" ]; then
      .      COMMAND="$COMMAND $SLOT=$slotVal"
    .    fi
  .done
  .COMMAND="$COMMAND $class_name $source"
  .
  .echo Forwarding event to $1 ...
  .eval $COMMAND
  .rc=$?
  .echo done
  .exit $rc
};

Task Send_Event {

  Description = (tec_tasks, "Send event to event server", 31);
  HelpMessage = (tec_tasks, "This task generates a new event and sends it to the selected event server. Select the server and specify the characteristics of the event to be sent.", 32);
  Roles = "super:senior:admin:user";

  Argument (tec_tasks, "Event server name", 33) {
    Layout = "EventServer";
    MustMatch = ".";
  }
};
Argument (tec_tasks, "message", 34) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "hostname", 35) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "origin", 36) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "severity", 37) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "sub_origin", 38) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "sub_source", 39) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "CLASS_NAME", 40) {
  Layout = "SimpleText";
};

Argument (tec_tasks, "SOURCE", 41) {
  Layout = "SimpleText";
};

Implementation ("default")
  .#!/bin/sh
  .echo Sending event to $1 ...
  .wpostemsg -S $1 msg=$2 hostname=$3 origin=$4 severity=$5 \
  sub_origin=$6 sub_source=$7 $8 $9
  .rc=$?
  .echo done
  .exit $rc
;}

Task Reset_Event_Log {
Description = (tec_tasks, "Reset transaction log", 42);

HelpMessage = (tec_tasks, "This task clears the transaction log of anything that is more than 30 seconds old.", 43);

Roles = "super:senior:admin:user";

Implementation ("default")
  .#!/bin/sh
  .echo Resetting event log ...
  .wtdbresetlog
  .rc=$?
  .echo done
  .exit $rc
;}

Task Clear_Reception_Log {
Description = (tec_tasks, "Clear reception log ", 132);
HelpMessage = (tec_tasks, "This task clears from the reception log anything that was received more than 30 seconds ago.", 133);

Roles = "super:senior:admin:user";

Implementation ("default")
  .!/bin/sh
  .echo Clearing reception log ...
  .wtdbclear -l -t 30
  .rc=$?
  .echo done
  .exit $rc

};

Task Clear_Closed_Events {

Description = (tec_tasks, "Clear closed events ", 44);

HelpMessage = (tec_tasks, "This task clears closed events from the event repository that are older than the specified time. Specify how old events should be to be cleared.", 45);

Roles = "super:senior:admin:user";

Argument (tec_tasks, "Seconds Old", 46) {
  Layout = "Number";
  ButtonLabel = (tec_tasks, "Seconds Old", 46);
  MustMatch = "^[0-9][0-9]*$";
  DefaultValue = "30";
};

Implementation ("default")
  .!/bin/sh
  .echo Clearing out closed events ...
  .wtdbclear -e -s CLOSED -t $1
  .rc=$?
  .echo done
  .exit $rc

};

Task Change_Severity {

Description = (tec_tasks, "Change severity of event", 47);

HelpMessage = (tec_tasks, "This task will change the severity of the current event. Select the new severity for the event.", 48);

Roles = "super:senior:admin:user";

Argument (tec_tasks, "Severity", 49) {
  Layout = "Severities";
  ButtonLabel = (tec_tasks, "Severity", 49);
  MustMatch = "^[A-Z]+$";
  DefaultValue = "FATAL";
};

Implementation ("default")
  .!/bin/sh
  .echo Changed event severity ...
  .wsetemsg -r $1 $CONSOLE_NAME $ev_key
  .rc=$?
  .echo done
  .exit $rc


Task Dump_Event_Repository {

Description = (tec_tasks, "Dump event repository", 50);

HelpMessage = (tec_tasks, "This task will dump out events from the event repository. Specify how many events to dump and whether they should be in ascending or descending order.", 51);

Roles = "super:senior:admin:user";

Argument (tec_tasks, "Ascending or Descending order?", 67) {
    Layout = "AscDesc";
};

Argument (tec_tasks, "Number of Events", 52) {
    Layout = "Number";
    ButtonLabel = (tec_tasks, "Number of Events", 52);
    MustMatch = "^[0-9]\[0-9]*$";
    DefaultValue = "1";
};

Implementation ("default")
    #!/bin/sh
    . echo Dumping $2 events from the event repository ...
    . wtdumper -d -o $1 -m $2
    . rc=$?
    . echo done
    . exit $rc
};

Task Backup_Event_log {

Description = (tec_tasks, "Backup the event log", 53);

HelpMessage = (tec_tasks, "This task will backup the event log. No input arguments are required.", 54);

Roles = "super:senior:admin:user";

Implementation ("default")
    #!/bin/sh
    . echo Backing up event log ...
    . wtdbbackup
    . rc=$?
    . echo done
    . exit $rc
};

Task Event_Query {

Description = (tec_tasks, "Query event repository", 109);

HelpMessage = (tec_tasks, "This task will perform a query on the event repository and display the events that match the query. Specify the values that you want to use to perform the query.", 110);

Roles = "super:senior:admin:user";

Argument (tec_tasks, "Match Class:", 120) {
    Layout = "SelectClass";
    DefaultValue = ";
};

Argument (tec_tasks, "Match Status:", 121) {
    Layout = "SelectStatus";
    DefaultValue = ";
};
Argument (tec_tasks, "Match Severity:", 122) {
  Layout = "SelectSeverity"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match Hostname:", 123) {
  Layout = "SelectHost"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match Source:", 124) {
  Layout = "SelectSource"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match SubSource:", 125) {
  Layout = "SimpleText"
  ButtonLabel = "SubSource"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match Origin:", 126) {
  Layout = "SimpleText"
  ButtonLabel = "Origin"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match SubOrigin:", 127) {
  Layout = "SimpleText"
  ButtonLabel = "SubOrigin"
  DefaultValue = "*"
};

Argument (tec_tasks, "Match Message:", 128) {
  Layout = "SimpleText"
  ButtonLabel = "Message"
  DefaultValue = "*"
};

Argument (tec_tasks, "Starting Date and Time:", 59) {
  Layout = "SimpleText"
  ButtonLabel = "Start Date"
  DefaultValue = "*"
};

Argument (tec_tasks, "Order Events by:", 116) {
  Layout = "Order"
  DefaultValue = "ASC"
  DefaultValue = "*"
};

Argument (tec_tasks, "Show values for slots:", 117) {
  Layout = "SelectSlots"
  DefaultValue = "*"
};

Implementation ("default")

#!/bin/sh

# This script is called by the event repository query task
# It formats some of the arguments into a where clause which
# can be passed to wtdumper. It also filters the output so
# that only interesting slots are actually printed.
#
# Functions

Chapter 10. Sample Task Library  469
.#
.make_in_char()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE="$COLUMN LIKE "\%" "
 . else
 . TEMP=`echo $VALUE | sed s/,/","/g`
 . VALUE="$COLUMN IN ($TEMP)"
 . fi
 .}

.make_in_int()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE=""
 . else
 . TEMP=`echo $VALUE | sed s/,/","/g`
 . VALUE="$COLUMN IN ($TEMP)"
 . fi
 .}

.make_like()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE="$COLUMN LIKE "\$VALUE" ="
 . else
 . VALUE="$COLUMN LIKE "\$VALUE" ="
 . fi
 .}

.# Mainline starts here
.#
.SCLASS=$1;shift
.STATUS=$1;shift
.SEVERITY=$1;shift
.HOSTNAME=$1;shift
.SOURCE=$1;shift
.SUB_SOURCE=$1;shift
.ORIGIN=$1;shift
.SUB_ORIGIN=$1;shift
.MSG=$1;shift
.START_TIME=$1;shift
.ORDER=$1;shift
.SLOTS=$1
.
.#
.# Format the where clause
.#

.make_in_char class "$CLASS"
.WHERE="$VALUE"

.STATUS=`echo "$STATUS" | sed -e s/OPEN/0/ \ 
 -e s/RESPONDED/10/ -e s/ACKNOWLEDGED/20/ -e s/CLOSED/30/` 
.make_in_int status "$STATUS"
.if [ "$VALUE" != "" ];then
 . WHERE="WHERE AND $VALUE"
 .fi

.SEVERITY=`echo "$SEVERITY" | sed -e s/UNKNOWN/10/ \
  -e s/HARMLESS/20/ -e s/WARNING/30/ \
  -e s/MINOR/40/ -e s/Critical/50/ \
  -e s/FATAL/60/`.

.make_int severity "$SEVERITY"

.if [ "$VALUE" != "" ];then
  WHERE="WHERE AND $VALUE"
.fi

.make_char hostname "$HOSTNAME"

.where="WHERE AND $VALUE"

.make_char source "$SOURCE"

.where="WHERE AND $VALUE"

.make_like sub_source "$SUB_SOURCE"

.where="WHERE AND $VALUE"

.make_like origin "$ORIGIN"

.where="WHERE AND $VALUE"

.make_like sub_origin "$SUB_ORIGIN"

.where="WHERE AND $VALUE"

.make_like msg "$MSG"

.where="WHERE AND $VALUE"

.# If slots are being specified.

.# First remove the list of slots to print from the

.# list of available slots. Then exclude these slots on

.# the output of the wtdumper command. This makes sure that

.# all slots that are extensions of the base event will print.

.if [ "$SLOTS" = -* -o "$SLOTS" = -- ];then
  if [ "$START_TIME" = "W" -o "$START_TIME" = "w" ];then
    wtdumper -d -o "$ORDER" -w "$WHERE"
  else
    wtdumper -d -t "$START_TIME" -o "$ORDER" -w "$WHERE"
  fi
.else
  SLOTS_TO_KEEP=`echo "$SLOTS" | sed -e s/"\$/-e / -e "s/,/\$/ -e "s/$/\$/"`
  SLOTS_TO_SKIP=`grep -v "$SLOTS_TO_KEEP" <<EOF
    server_handle
date_reception
event_handle
source
sub_source
origin
sub_origin
hostname
adapter_host$status
administrator
acl
severity
date
duration
msg
msg_catalog
msg_index
num_actions
credibility
repeat_count
cause_date_reception
cause_event_handle
EOF`
Chapter 10. Sample Task Library 471
SLOTS_TO_SKIP=`echo $SLOTS_TO_SKIP \ 
    | sed -e "s/=/ -e /g" -e "s/^/ -e /" -e "s/$=/"`

.#
.# Run the wtdumper command and pipe it to the output formatter
.#
.# if [ "$START_TIME" = "*" -o "$START_TIME" = "" ];then
    wtdumper -d -o "$ORDER" -w "$WHERE" \ 
    | grep -v $SLOTS_TO_SKIP
. else
    wtdumper -d -t "$START_TIME" -o "$ORDER" -w "$WHERE" \ 
    | grep -v $SLOTS_TO_SKIP
. fi
. fi

.exit $?;

};

Task page_admin {

Description = (tec_tasks, "Send alphanumeric page", 60);

HelpMessage = (tec_tasks, "This task will use the fbeep program to send a page. The device name for the modem, phone number, page number and the message number must be supplied as input arguments. It is assumed that the fbeep program can be found in the search path. If you do not have this program the source for it is shipped with IBM Tivoli Enterprise Console and it gets installed in $BINDIR/TME/TEC/contrib/pager.", 61);

Argument (tec_tasks, "Device", 62) {
    Layout = "SimpleText";
    DefaultValue = "/dev/ttya";
    MustMatch = ".";
};

Argument (tec_tasks, "Phone Number", 63) {
    Layout = "SimpleText";
    DefaultValue = "9,18005551000";
    MustMatch = ".";
};

Argument (tec_tasks, "Pager Number", 64) {
    Layout = "SimpleText";
    DefaultValue = "1234567";
    MustMatch = ".";
};

Argument (tec_tasks, "Message", 65) {
    Layout = "SimpleText";
    MustMatch = ".";
};

Roles = "super:senior:admin:user";

Implementation ("default")
cheiden -xw
    .fbep -vv -D $1 -d $2 -p $3 - $4
    .rc=$?
    .echo Page sent with return code $rc.
    .exit $rc
    ;
};
Task Find_Similar_Events {

Description = (tec_tasks, "Find matching event", 118);

HelpMessage = (tec_tasks, "This task will perform a query to find events that have similar characteristics as the task in whose context this task is executed. Specify which attributes you want to match on.", 119);

Roles = "super:senior:admin:user";

Argument (tec_tasks, "Match Class:", 120) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Status:", 121) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Severity:", 122) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Hostname:", 123) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Source:", 124) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match SubSource:", 125) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Origin:", 126) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match SubOrigin:", 127) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Match Message:", 128) {  
  Layout = "YesNo";
  DefaultValue = "YES";
};

Argument (tec_tasks, "Starting Date and Time:", 59) {  
  Layout = "SimpleText";
  DefaultValue = "*";
};

Argument (tec_tasks, "Order Events by:", 116) {  
  Layout = "Order";
  DefaultValue = "ASC";
};

Argument (tec_tasks, "Show values for slots:", 117) {  
  Layout = "SelectSlots";
};
DefaultValue = "*";
};

Implementation ("default")
.#!/bin/sh
.#
.## This script is called by the event repository query task
.## It formats some of the arguments into a where clause which
.## can be passed to wtdumper. It also filters the output so
.## that only interesting slots are actually printed.
.##
.## Functions
.##
.make_in_char()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE="$COLUMN LIKE \"%\" "
 . else
 . TEMP=`echo $VALUE | sed s/,/","/g`
 . VALUE="$COLUMN IN (\"$TEMP\")"
 . fi
 .}

.make_in_int()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE=""
 . else
 . TEMP=`echo $VALUE | sed s/,/","/g`
 . VALUE="$COLUMN IN ($TEMP)"
 . fi
 .}

.make_like()
.{
 . VALUE=$2
 . COLUMN=$1
 . if [ "$VALUE" = "*" -o "$VALUE" = "" ]; then
 . VALUE="$COLUMN LIKE \"%\" "
 . else
 . VALUE="$COLUMN LIKE \"$VALUE\" "
 . fi
 .}

.# Mainline starts here
.#
.if [ "$1" = "NO" ];then
 . CLASS="*"
 .else
 . CLASS="$class"
 .fi
 .shift
.if [ "$1" = "NO" ];then
 . STATUS="*"
 .else
 . STATUS="$status"
 .fi
 .shift
.if [ "$1" = "NO" ];then
 . SEVERITY="*"
 .else
 . SEVERITY="$severity"
.fi
.shift
.if [ "$1" = "NO" ];then  
  HOSTNAME="*"
.else  
  HOSTNAME="$hostname"
.fi
.shift
.if [ "$1" = "NO" ];then  
  SOURCE="*"
.else  
  SOURCE="$source"
.fi
.shift
.if [ "$1" = "NO" ];then  
  SUB_SOURCE="*"
.else  
  SUB_SOURCE="$sub_source"
.fi
.shift
.if [ "$1" = "NO" ];then  
  ORIGIN="*"
.else  
  ORIGIN="$origin"
.fi
.shift
.if [ "$1" = "NO" ];then  
  SUB_ORIGIN="*"
.else  
  SUB_ORIGIN="$sub_source"
.fi
.shift
.if [ "$1" = "NO" ];then  
  MSG="*"
.else  
  MSG="$msg"
.fi
.shift
.START_TIME=$1;shift
.ORDER=$1;shift
.SLOTS=$1
.
.
# Format the where clause
#
.
.make_in_char class "$CLASS"
.WHERE="$VALUE"
.
.STATUS=`echo "$STATUS" | sed -e s/OPEN/0/ \  
  -e s/RESPONDED/10/ \  
  -e s/ACKNOWLEDGED/20/ \  
  -e s/CLOSED/30/ ` 
.make_in_int status "$STATUS"
.if [ "$VALUE" != "" ];then  
  WHERE="$WHERE AND $VALUE"
.fi
.
.SEVERITY=`echo "$SEVERITY" | sed -e s/UNKNOWN/10/ \  
  -e s/HARMLESS/20/ \  
  -e s/WARNING/30/ \  
  -e s/MINOR/40/ \  
  -e s/CRITICAL/50/ \  
  -e s/FATAL/60/ ` 
.make_in_int severity "$SEVERITY"
.if [ "$VALUE" != "" ];then  
  WHERE="$WHERE AND $VALUE"
.fi
.fi
.
.make_in_char hostname "$HOSTNAME"
.WHERE="$WHERE AND $VALUE"
.
.make_in_char source "$SOURCE"
.WHERE="$WHERE AND $VALUE"
.
.make_like sub_source "$SUB_SOURCE"
.WHERE="$WHERE AND $VALUE"
.
.make_like origin "$ORIGIN"
.WHERE="$WHERE AND $VALUE"
.
.make_like sub_origin "$SUB_ORIGIN"
.WHERE="$WHERE AND $VALUE"
.
.make_like msg "$MSG"
.WHERE="$WHERE AND $VALUE"
.
#
.# Logic when slots are being specified.
.# Remove the list of slots to print from the
.# list of available slots. Then exclude the slots on
.# the output of the wtdumper command. This makes sure that
.# all slots that are extensions of the base event will print.
.#
.
.if [ "$SLOTS" = "*" -o "$SLOTS" = "" ];then
  
  # Run the wtdumper command
  
  # if [ "$START_TIME" = "" -o "$START_TIME" = "*" ];then
  wtdumper -d -o "$ORDER" -w "$WHERE"
  else
  wtdumper -d -t "$START_TIME" -o "$ORDER" -w "$WHERE"
  fi
.endif
.
.else
  SLOTS_TO_KEEP=`echo $SLOTS \
  | sed -e "s/\-/\-e "s/\,/\$ -e /g" -e "s/$\$/\$/"
  SLOTS_TO_SKIP=`grep -v $SLOTS_TO_KEEP <<EOF
  server_handle
  date_reception
  event_handle
  source
  sub_source
  origin
  sub_origin
  hostname
  adapter_host
  status
  administrator
  acl
  severity
  date
  duration
  msg
  msg_catalog
  msg_index
  num_actions
  credibility
  repeat_count
  cause_date_reception
  cause_event_handle
  EOF`
  
  SLOTS_TO_SKIP="echo $SLOTS_TO_SKIP \
  | sed -e "$SLOTS_TO_SKIP "s/\-/\-e "s/\,/\$ -e /g" -e "s/$\$/\$/"
  
 EOF"
sed -e "s/ /= -e "s/^/-e "s/$=/`

# Run wtdumper and pipe it to the output formatter
# if [ "$START_TIME" = ";0= $START_TIME" = ";0= ]
    wtdumper -d -o "$ORDER" -w "$WHERE" \
    | grep -v $SLOTS_TO_SKIP
  else
    wtdumper -d -t "$START_TIME" -o "$ORDER" -w "$WHERE" \
    | grep -v $SLOTS_TO_SKIP
  fi
  fi

exit $?

Task Clean_Database {
Description = (tec_tasks, "Clean out event repository", 129);
HelpMessage = (tec_tasks, "Clean out old events from the event repository.", 130);
Roles = "super:senior:admin:user";
Implementation ("default")
  #! /bin/sh
  .. /etc/Tivoli/setup_env.sh
  ..
  
  .TNR=${TNR:-`wlookup NameRegistry`}
  .export TNR
  .methodpath="$EXECDIR/$INTERP/TME/TEC"
  .export methodpath
  .EventServer=`wlookup -r EventServer EventServer`
  .wtdbresetlog

  .REC_LOG_TIME=`idlattr -t -g $EventServer recv_log_time ulong`
  .wtdbclear -l -t $REC_LOG_TIME 1> /dev/null 2>&1
  .if [ $? -ne 0 ]
    then
      echo "There was a problem executing wtdbclear, ret = $?"
      exit 1;
    fi
  ..

  .CLOSED_CACHE_TIME=`idlattr -t -g $EventServer rule_cache_full_history ulong`
  .wtdbclear -e -s CLOSED -t $CLOSED_CACHE_TIME 1> /dev/null 2>&1
  .if [ $? -ne 0 ]
    then
      echo "There was a problem executing wtdbclear, ret = $?"
      exit 1;
Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

---

**Trademarks**

AIX, AS/400, DB2, IBM, MVS, OS/2, OS/400, S/390, Tivoli, Tivoli Enterprise, Tivoli Enterprise Console, TME, TME 10, WebSphere, WIN-OS/2, and z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. For a complete list of Intel trademarks, see [http://www.intel.com/sites/corporate/tradmarx.htm](http://www.intel.com/sites/corporate/tradmarx.htm).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.
Index

Special characters
- .odbc.ini file 135, 338
- .tivoli_rc file 371

A
- access, remote 245
  Windows endpoints 245
  Windows managed nodes 361
- accessibility ix
- administrator commands
  table of 7
  wauthadmin 80
  wcdr 90
  wcrtdadmin 121
  wgetadmin 202
  widmap 226
  wpwd 296
  wsetadmin 345
- wsetlang 351
- administrators
  changing group roles 345
  changing logins 345
  changing notice group subscriptions 345
  creating 121
  creating root administrators 80
  deleting root administrators 80
  displaying working collection 296
- retrieving information 202
- after_install_policy method 402
- ALI value 440
- allow_install_policy method 398
- Application Developer's Environment 436
- applications
  installing 234
  uninstalling 385
- arg file 456
- ArgLayout keyword 441
- Argument clause 454
- Argument layouts 441
- attributes
  ButtonLabel 454
  Comments 451
  Context 440
  DefaultValue 454
  Description 451
  Distribution 440
  Gid 451
  HelpMessage 440, 451
  MustMatch 454
  MustNotMatch 454
  Requires 440
  Roles 452
  SectionLabel 454
  Uid 452
  Version 440
- attributes, implementation
  adding 17
  getting 17
  setting 17

B
- batch file commands 348
- Binary interp 452
- bold face type, meaning of ix books
  See publications
- boot methods, for endpoints 171
- ButtonLabel attribute 454
- ButtonLabel keyword 441

C
- C preprocessor 436
- Choice layout 443
- ChoiceButton layout 447
- code sets, converting 224
- collections
  changing current working 90
  displaying current working 296
  linking objects 247
  listing information 254
  listing member objects 254
  moving objects 286
  refreshing 305
  removing objects from 315
- command table
  administrator 7
  configuration management 7
  endpoint 7
  gateway 7
  httpd 8
  installation 9
  interregion 9
  low-level maintenance 9
  managed node 10
- miscellaneous 13
- multiplexed distribution 10
- notification 11
- policy 11
- query 12
- RDBMS Interface Module 12
- Revision Control System 12
- scheduler 12
- task library 13
- commands
  wcrtpol 429
  wgteppol 429
  wgetpol 429
  wslpol 429
  wsllpol 429
  wputeppol 429
  wputpol 429
- commands, provided by Tivoli Management Framework 1
- Comments attribute 451
- communication, for DB2 136
- configuration management commands
  table of 7
  wcrtpol 129
  wcrtpolmgr 131
  wdistrib 160
  wgetpol 213
  wgetsub 218
  wlsuser 265
  wpopulate 291
  wsetpm 354
  wsuser 370
  wuninst 385
  wunsub 389
  wvalidate 392
- connections, Tivoli region
  exchanging information 255
  listing 255
- Context attribute 440
- cpp command 436
- CryptoText layout 441
- custom dialogs
  merging 284
  saving 88
- customer support
  See software support

D
- database ID, Informix 135, 338
- database mode, for profile managers 354
- database server ID, Informix 137
- datless mode, for profile managers 354
- date command 142
- DB2
  communication 136
  DB2COMM variable 136
  DB2INSTANCE variable 137
  DBDIR variable 136
  installation directory 136
  instance home 136
  instance name 137
  instance owner 137
  INSTHOME variable 136
- DB2COMM variable 136
- DB2DIR variable 136
- DB2INSTANCE variable 137
- default files 452
- default policies
  association with policy regions 355
  listing policy methods 263
  setting defaults for object classes 347
- DefaultValue attribute 454
- dependencies, setting 437
- dependency sets
  associating 91
  creating 155
  editing 155
  resolving 155
  viewing 155

© Copyright IBM Corp. 2003
E-mail
configuring for Windows 267
SMTP 267
specifying server 267
endpoint and gateway commands
table of 7
waddpath 78
wdkskc 164
weditini 166
wep 171
wepmgr 181
wesstatus 186
wepupgd 189
wgateway 192
wgetkey 209
wgetval 220
winsblk 230
winsline 232
winstlcf 238
wleendpts 237
wmgini 285
wrestart 308
wrlblk 318
wrline 319
wseterr 348
wsetval 364
endpoint commands
lcfd 25
lcfd.sh 35
w4inslcf.pl 73
wadminep 79
wcrlblk 108
wcrline 109
wcryptfile 120
wdelap 145
wdelgate 146
wlfcatp 245
endpoint manager
configuring 181
controlling 181
endpoint policy
about 397
endpoint policy (continued)
after_install_policy 402
allow_install_policy 399
editing 429
enabling automatic software update 405
login_policy 405
select_gateway_policy 407
table of 397
where and when it runs 397
endpoint policy scripts, getting 206
endpoints
automatically upgrading on log in 405
configuring 25
default list in task libraries 421
deleting 145
determining gateways to manage 407
initial post installation processing 402
installing 238
installing (AS/400) 73
installing previous version 189
listing subscription to profile manager 257
login processing 405
migrating 171
setting HTTP password 171
setting logging options 25
setting login options 25
setting properties 171
setting remote access account (Windows) 245
starting UNIX endpoints 35
status 186
stopping UNIX endpoints 35
upgrading 189
using batch files 348
validating as task targets 425
validating connection 398
viewing information 171
waking up 79
working with boot methods 171
environment variables
See variables
files
(continued)
logs command 36
merging three-ways 283
modified during managed node installation 103
odtrace.log 381
readable transactions logs 36
setup_env 3
sup.SUNAME. uu 371
task 456
temporary 377
tsnames.ora 137
wsupport.SUNAME 371
files, text
inserting blocks 230
inserting lines 232
removing blocks 108
removing lines 109
replacing blocks 318
replacing lines 319
G
gateway commands
See also endpoint and gateway commands
wcreate 123
wdeploy 146
gateway HTTP server, controlling access 192
gateways
configuring 192
controlling gateway HTTP server 192
creating 123
defining logging characteristics 192
deleting 146
determining endpoints to manage 407
listing properties 192
starting 192
stopping 192
Gid attribute 451
GLOBAL value 440
H
headers
information 439
task library 439
HelpMessage attribute 440, 451
home directory, DB2 instance 136
host
e-mail 267
SMTP server 267
host name, setting in Windows registry 249
HostFileBrowser keyword 444
hostid command 221
HTTP
department listings 222
forwarding requests 222
gateway listings 222
HTTP server, gateway 192
httpd commands
table of 8
httpd commands (continued)
whttpd 222

I

dragging to Windows Program Manager 76
IDL commands
idlargs 16
idlattr 17
idcall 19
idlexception 21
idlinput 23
idrisult 24
invoking from shell 19
idlarg command 16
idlattr command 17
idcall command 19
idlexception command 21
idlinput command 23
idresult command 24
ifconfig command 228
Implementation 452
implementation attributes
adding 17
getting 17
setting 17
Informix
odbmi.ini file 135, 358
database ID 135, 358
database server ID 137
informix account 137
INFORMIXDIR variable 136
INFORMIXSERVER variable 137
installation directory 136
INFORMIXDIR variable 136
INFORMIXSERVER variable 137
INI files
merging contents 285
modifying 166
installation commands
oinstall 61
table of 9
wclient 103
wpcdm from 118
winstall 234
wstcl 238
wmailhost 267
wp 289
wpset 339
wsset 361
wstandalone 367
installation directory
DB 2 136
getting for managed nodes 236
Informix 136
installation images, copying for CD 118
installation, upgrading products 289
instance home, DB 2 136
instance name, DB 2 137
instance name, Oracle 135, 358
instance owner, DB 2 137
INSTHOME variable 136
interfaces file, Sybase 136
interfaces, IP (continued)
adding to managed nodes 228
interfaces, IP (continued)
changing properties for managed nodes 228
listing for managed nodes 228
removing from managed nodes 228
internationalization
converting characters between code sets 224
finding localized files and directories 251
interpreter types
getting for managed nodes 244
introduction vii
intersection commands
table of 9
wconnect 115
wsconnect 158
wlook up 252
wsconnect 255
wreg 256
wtsname 380
wuwipdate 391
inventory information, querying 327
IP interfaces
adding to managed nodes 228
changing properties for managed nodes 228
listing for managed nodes 228
removing from managed nodes 228
italic typeface, meaning of ix

J
jobs
creating 125
deleting for task library 147
editing scheduled 167
enabling scheduler 170
listing properties 208
listing within task libraries 266
removing from scheduler 150
retrieving scheduling for 216
running 329
scheduling 336
setting properties 349
validating endpoint targets 425
validating group ID 427
validating managed node targets 425
validating targets in profile managers 426
validating user ID 428

K
keywords (continued)
task 451

L
LANG environment variable 86, 251
layout file 456
Layout keyword 454
layouts
Choice 443
ChoiceButton 447
CryptoText 441
Text 441
TextChoice 443
layouts, arguments 441
lcf command 25
lcf.sh command 35
legal notices 479
library file 456
listener service, Oracle 137
LOCAL value 440
locale, setting 351
localization, finding files and directories 251
locks, transactions 70
log files
logs command 36
odtrace.log 381
transactions, readable 36
log messages, Revision Control System 312
login_policy method 405
logs command 36
Lotus Notes, as mail server 267
low-level maintenance commands
idlargs 16
idlattr 17
idcall 19
idlexception 21
idlinput 23
idresult 24
logs 36
objc 38
odadmin 41
odbs 54
odstat 56
oserv 63
table of 9
tmcmd 69
tmstat 70
wlocalhost 249
wlocktmr 250
wmailhost 267

M
mail server
for Tivoli tools 267
specifying for Windows 267
maintenance commands, low-level
maintenance, putting Tivoli region in maintenance mode 250
managed node commands

managed node commands (continued)

wchknode 94
wclient 103
wdate 142
wdirs 159
whostid 221
wifconfig 228
winstdir 236
winterp 244
wmannode 268
wmernsize 282
wstandalone 367
wtimezone 376
wuname 384
wunstmn 387
wxterm 393

managed nodes

adding IP interfaces 228
backing up object databases 282
restoring object databases 282
changing IP interface properties 228
checking status 72
configuring as repeaters 320
default list in task libraries 421
determining memory 282
getting environment 41
getting installation directory 236
getting interpreter type 244
getting system information 268
installation, files modified 103
installing 367
listing information 41
listing IP interfaces 228
listing operating system information 384
making repeaters 320
opening an Xterminal window 393
printing 72
printing date and time 142
printing host ID 221
printing time zone 376
providing list of subscribers to profile managers 414
recycling 41
removing 316, 387
removing IP interfaces 228
setting 41
setting locale 351
setting remote access account
(Windows) 361
starting Distribution Status console 281
synchronizing object databases 41
uninstalling 387
updating database references 94
validating as task targets 425
verifying database references 94
managed resource types
adding to policy regions 355
removing from policy regions 355
managed resources
listing 252
listing policy methods 263
listing policy objects 262
searching name registry 252

manus

See publications
maps, user logins 226
MDist 2 distributions
configuring for multicast 269
configuring repeaters 272
managing 272
managing repeater depots 152
message catalog formats 440
message tables 436
messages
binding variables 86
broadcasting to Tivoli desktop 87
localization 86
retrieving locale-specific 86
method calls, tracing 381
methods
boot methods for endpoints 171
endpoint policy
after_install_policy 402
allow_install_policy 398
login_policy 405
select_gateway_policy 407
formatting output 24
profile manager default policy
pm_def_profile_manager 412
pm_def_profile_types 413
pm_def_subscribers 414
profile manager validation policy
pm_val_remove_subscribers 415
pm_val_remove_subscription 416
pm_val_subscribers 417
pm_val_subscription 418
run_task 437
running as object calls 38
setuid 97
shell 24
generating input options 23
raising exceptions 21
task library default policy
tl_def_dist_mode 420
tl_def_man_nodes 421
tl_def_prof_mgrs 422
tl_def_set_uid 424
task library validation policy
tl_val_man_nodes 425
tl_val_prof_mgrs 426
tl_val_set_uid 428
tl_val_set_gid 423, 427
Microsoft Exchange, as mail server 267
miscellaneous commands

table of 13
tivoli 67
tivoli 72
vdisp 72
vaddicon 76
vbindmsg 86
vbkupdb 82
vcatcher 88
wcd 90
wchdep 91
wchkbd 92
wchknode 94
welde 143
wddepot 152
wdsed 155
wdsmsg 165
wgetallinst 204

miscellaneous commands (continued)
wiconv 224
winstednt 237
wlfctap 245
wln 247
wlocpath 251
wlist 254
wlnst 258
wmcast 269
wmdist 272
wmdist 281
wmerge 283
wmrqae 284
wmt 286
wping 290
wpm 296
wrefesh 305
wrm 315
wrmnode 316
wrpt 320
wrmas 326
wsed 353
wsed 371
wtemp 377
wtemp 381
monospace font, meaning of ix
Multi keyword 443
multicast distributions, configuring 269
multiplexed distribution commands, table of 10
MustMatch attribute 454
MustNotMatch attribute 454

N

name registry

across connected Tivoli regions 391
listing resources 252
searching for resources 252
updating resources 391
name, DB2 instance 137
NetWare configuration programs, copying 120
NLSPATH environment variables 86
NLS_PATH variable 251
notice groups
changing subscription 345
expiring notices 191
listing 260
notices 479
displaying 373
expiring 191
filtering 373
listing 260
sending 365
notification commands

table of 11
wbroadcast 87
wesnotif 191
wlsnotif 260
wsnotif 365
wtailnotif 373
ordering ORACLE_HOME operating online odstat odadmin objects objectcall command 38 object calls, tracing 381 object databases backing up 82 deleting objects 143 listing contents 54 local 389
low-level administration 143 repairing 92 restoring 82 synchronizing 41 updating local with server 94 verifying 92 verifying local with server 94 object dispatchers allowing managed node installation 41 configuring 63 controlling 63 enabling tracing 381 getting environment 41 initializing 63 installing service in Windows Service Manager 61 listing information 41 pinging 290 recycling 41 removing service in Windows Service Manager 61 setting 41 setting encryption levels 41 setting network security 41 starting 41, 63 stopping 41 synchronizing object databases 41 updating service in Windows Service Manager 61 objects
DependencyMgr 91, 155
listing members in collections 254 moving between collections 286 removing from collections 315
odadmin command 41
odbis command 54
odstat command 56
oinstall command 61
online publications accessing viii
operating system information, listing 384
operating systems, restarting (Windows) 308
Oracle
instance name 135, 358
listener service 137
ORACLE_HOME variable 136 ORACLE_SID variable 135, 358 system ID 135, 358 tnsnames.ora file 137 TWO_TASK variable 137 ORACLE_HOME variable 136 ORACLE_SID variable 135, 358 ordering publications ix oserv command 63 out-of-cache dependencies 155

password key, in Window registry 353 passwords 441
Patches
installing 289
listing installed 258
ping command 290
platform commands 7
pm_def_profile_managers method 412
pm_def_profile_types method 413
pm_def_subscribers method 414
pm_val_remove_subscribers method 415
pm_val_remove_subscription method 416
pm_val_subscribers method 417
pm_val_subscription method 418
policies
creating policy objects 127
setting for object classes 347
verifying members 96
policy commands
table of 11
wchkpol 96
wcrtpol 127
wcrtrp 128
wdelpol 148
wdepr 149
wgdftp 205
wgttteppol 206
wgttep 206
wgttepom 210
wgttr 212
wlspol 262
wlspolm 263
wpupeppol 292
wpuptom 293
wseftp 347
wsepr 355
policy methods
See also endpoint policy, task library policy, profile manager policy listing 263
listing contents 210
replacing body 293
policy objects
creating 127
deleting 148
listing 205, 262
settings defaults 347
policy regions
adding resource types 355
assigning policies 355
creating 128
creating query libraries 132
creating task libraries 141
default policies 355
deleting 149
providing list of profile types 413
removing resource types 355
retrieving information 212
setting policy object defaults 347
validation policies 355
verifying policies 96
policy scripts
endpoint, replacing 292
endpoints 206
policy verification 96
problem diagnosis 381
products
installing 234
listing installed 258
uninstalling 385
upgrading 289
profile manager policy about 411
default policy methods
pm_def_profile_managers 412
pm_def_profile_types 413
pm_def_subscribers 414
table of 411
describing 429
validation policy methods
pm_val_remove_subscribers 415
pm_val_remove_subscription 416
pm_subscribers 417
pm_val_subscription 418
table of 411
profile managers
creating 131
default list for task libraries 422
disabling database mode 354
disabling dataloss mode 354
enabling database mode 354
enabling dataloss mode 354
listing endpoint subscribers 257
listing subscribers 218, 265
providing list of manager node subscribers 414
providing profile list 412
removing subscribers from profile managers 389
retrieving profile copies 213
subscribing resources 370
unsuscribing resources 389
validating job targets 426
validating profile manager subscriptions 418
validating removal of subscribers 415
validating subscribers 417
validating subscription cancellation 416
validating task targets 426
profiles
cloning 129
concatenating 291
distributing 160
listing attributes 263
overwriting 291
populating 291
providing list for profile managers 412
providing list of types for policy regions 413
retrieving subscription copies 213
validating 392
Program keyword 444, 447, 449
ProgramData keyword 444, 447, 449
publications
accessing online viii
ordering ix
pwd command 296
Index 487
Resource types (continued)
removing from policy regions 355
ResourceOID keyword 444
resources
defined in policy region 212
deleting from object database 143
exchanging across Tivoli regions 306
removing subscribers from profile
managers 389
subscribing to profile managers 370
unsubscribing from profile
managers 389
updating in connected Tivoli
regions 391
return codes, setting for batch files 348
Revision Control System
changing file attributes 297
checking in revisions 97
checking out files 110
comparing files 301
creating RCS files 297
merging files 303
printing log messages 312
searching for patterns 225
Revision Control System commands
table of 12
wci 97
wco 110
wident 225
wrcs 297
wrcsdiff 301
wrcsmerge 303
wrlog 312
RIM objects
changing password 360
creating 135
editing properties 358
moving 287
retrieving information 215
setting password 360
testing 309
tracing 310
Roles attribute 452
root administrators
granting authorization 80
listing 80
removing authorization 80
RSTLCPGM command (AS/400) 73
RSTOBJ command (AS/400) 73
select_gateway_policy method 407
service, Oracle listener 137
setuid methods 97
setup_env.csh file 3
setup_env.sh file 3
shell method
raising exceptions 21
shell methods
formatting output 24
getting input options 23
SMTP
connecting to Windows 267
gateway software 267
mail server 267
software support
contacting ix, 371
gathering information for 371
spider service 222
status, for endpoints 186
STRTMEET command (AS/400) 73
subscribers
listing 218, 265
removing 389
subscription list 389
subscriptions, retrieving profile
copies 213
support
See software support
Sybase
DSQUERY variable 137
interfaces file 136
SYBASE variable 136
SYBASE variable 136
system ID, for Oracle 135, 358
T

tar command 435
task binaries, controlling
distributions 162
task definitions 451
task file 456
task headers 451
task keyword 451
task libraries
creating 141, 378
creating tasks 139
default group ID for tasks 423
default list of endpoints 421
default list of managed node 421
default list of profile managers 422
default user ID for tasks 424
deleting jobs 147
deleting tasks 151
listing tasks and jobs 266
modifying 378
validating group ID for jobs 427
validating group ID for tasks 427
validating user ID for jobs 428
validating user ID for tasks 428
task library
definition 435
headers 439
task library commands
table of 13
wcrjob 125
wcrrtask 139
task library commands (continued)
  wcrtlib 141
  wdejob 147
  wdeltask 151
  wdlstask 162
  wgejob 208
  wgettask 219
  wslib 266
  wrunjob 329
  wruntask 333
  wsetjob 349
  wsettask 363
  wtaskabort 374
  wtll 378

task library definitions
  exporting 378
  importing 378

task library policy
  about 429
  default policy methods
    table of 419
    tl_def_dist_mode 420
    tl_def_man_nodes 421
    tl_def_prof_mgrs 422
    tl_def_set_uid 424
  editing 429
  validation policy methods
    table of 419
    tl_val_man_nodes 425
    tl_val_prof_mgrs 426
    tl_val_set_gid 423, 427
    tl_val_set_uid 428

task transactions
  canceling 374
  rolling back 374
  stopping 374
  TASK_BINDIR variable 452
  TaskLibrary keyword 439

tasks
  binaries, default distribution
    mode 420
    creating 139
  default group ID 423
  default user ID 424
  defining 451
  definition 435
  deleting 151
  distributing binaries 162
  enabling scheduler 170
  listing properties 219
  listing within task libraries 266
  running 333
  setting properties 363
  validating endpoint targets 425
  validating group ID 427
  validating managed node targets 425
  validating targets in profile
    managers 426
  validating user ID 428

tasks libraries
  querying distribution mode 162
  setting distribution mode 162
  temporary directory, validating 377
  text files
    inserting blocks 230
    inserting lines 232
    removing blocks 108
  text files (continued)
    removing lines 109
    replacing blocks 318
    replacing lines 319
  Text layout 441
  TextChoice layout 443
  time zones, determining 376
  Tivoli Authentication Package
    setting for Windows endpoints 245
    setting for Windows managed
      nodes 361
  tivoli command 67
  Tivoli databases
    backing up 82
    restoring 82
  Tivoli desktop
    broadcasting messages 87
    customizing startup message 165
    starting 67
  Tivoli desktop, refreshing collections 305
  Tivoli object database
    repairing 92
    verifying 92
  Tivoli region
    changing name 380
    connecting 115
    displaying name 380
    exchanging connection
      information 255
    exchanging resources 306
    listing connections 255
    performing remote operations 41
    placing in maintenance mode 250
    setting encryption levels 41
    validating endpoint connection 398
  Tivoli regions
    backing up object databases
      restoring object databases 82
      disconnecting 158
  Tivoli remote access account
    for Windows endpoints 245, 361
  Tivoli server
    allowing managed node
      installation 41
    getting environment 41
    installation (UNIX) 339
    listing information 41
    recycling 41
    setting 41
    setting locale 351
    synchronizing object databases 41
  Tivoli software information center viii
  TIVOLI_COMM_DIR variable 106, 342
  tl_def_dist_mode method 420
  tl_def_man_nodes method 421
  tl_def_prof_mgrs method 422
  tl_def_set_uid method 424
  tl_val_man_nodes method 425
  tl_val_prof_mgrs method 426
  tl_val_set_gid method 423, 427
  tl_val_set_uid method 428
  tlconf directory 456
  tmcmd command 69
  TME 10 ADE 436
  TMFDIR variable 97
  tmstat command 70
  tnsnames.ora file 137
  TOOLS variable 438
  transaction manager 69
  transactions
    displaying current 70
    displaying locks 70
    forcing change 69
    hierarchies 6
    logs command 36
    readable log files 36
    revocable subtransactions 6
    subtransactions 6
  transactions, tasks
    canceling 374
    rolling back 374
    stopping 374
  TWO_TASK variable 137
  typeface conventions ix

U
  Uid attribute 452
  upgrade.sh command 405
  user accounts
    informix 137
    user logsins maps 226

V
  validation policies
    association with policy regions 355
    listing policy methods 263
    setting defaults for object classes 347
    validating profiles 392
  values
    ALI 440
    GLOBAL 440
    LOCAL 440
  variables
    DB2COMM 136
    DB2DIR 136
    DB2INSTANCE 137
    DSQUERY 137
    INFORMIXDIR 136, 137
    INSTHOME 136
    LANG 86
    NLSPATH 86
    ORACLE_HOME 136
    ORACLE_SID 135, 358
    setting 3
    SYBASE 136
    TIVOLI_COMM_DIR 106, 342
    TMFDIR 97
    TOOLS 438
    TWO_TASK 137
  variables, notation for ix
  vdisp command 72
  Version attribute 440

W
  w4inslcf.pl command 73
  waddicon command 76
  waddpath command 78
  wadminp command 79
  Wake on LAN 33
  wauthadmin command 80